Parallelised Scalable Simulations of Biological Neural Networks using TensorFlow: A Beginners' Guide

Saptarshi Soham Mohanta[1], Collins Assisi[1], with Theoretical Neuroscience Group¶.

1 Indian Institute of Science Education and Research, Pune, Maharashtra, India

¤Current Address: Biology Department, Indian Institute of Science Education and Research, Pune, Maharashtra, India
¶Membership list can be found in the Acknowledgments section.
* collins@iiserpune.ac.in

Abstract

The dynamics of neurons and their networks have been studied extensively by modelling them as collections of differential equations, and the power of these mathematical tools are well recognised. Many tools and packages exist that allow the simulation of systems of neurons using these differential equations. However, there is a barrier of entry in terms of developing flexible general purpose simulations that are platform independent and support hardware acceleration on modern computing architectures such as GPUs/TPUs and Distributed Platforms. TensorFlow is a Python-based open-source package initially designed for machine learning algorithms, but it presents a scalable environment for a variety of computation including solving differential equations using iterative algorithms from numerical analysis such as Runge-Kutta methods. There are two significant benefits of such an implementation: high readability and scalability across a variety of computational devices. We explore the process of implementing a scalable simulation of a system of neurons based on Hodgkin-Huxley-like neuron equations using TensorFlow. We also discuss the limitations of such implementation and approaches to deal with them.

Author summary

In the form of a 7-day tutorial, the reader is introduced to the mathematical modelling of neuronal networks based on the Hodgkin-Huxley Differential Equations and is instructed on developing highly parallelised but easily readable code for numerical methods such as Euler's Method and Runge-Kutta Methods to solve differential equations in Python and use them to simulate neuronal dynamics. To develop scalable code, Google's open-source package TensorFlow is introduced, and the reader is instructed in developing simulations using this package and handling the few limitations that come with this implementation. The reader is also introduced to coding structures that maximise the parallelizability of the simulation. Finally, the coding paradigm that was developed is used to simulate a model of Locus Antennal Lobe described in previous literature, and its efficacy is analysed.

Introduction

The processing of information by the nervous system spans across space and time, and mathematical modelling of these dynamics have found to be an essential tool. These models have been used extensively to study the dynamics and mechanisms of information processing at both the individual neuron level and the system of neurons level. These models generally utilise systems of simultaneous ordinary differential equations (ODEs) which are solved as initial value problems using well-studied methods from numerical analysis such as Euler's Method and Runge Kutta methods. From a detailed study of the mechanism of action of the neurons, ion channels, neurotransmitters or neuromodulators and their dynamics in different models, equations have been found that describe the behaviour of neurons and synapse. By forming interconnected systems of these individual groups of differential equations, the overall dynamics and behaviour of networks can be studied through deterministic or stochastic simulations which can be easily perturbed unlike the case for in vivo experiments.

A significant issue with such simulations is computational complexity. As the number of neurons increase, the number of possible synaptic connections increases quadratically. That is, for a system of n neurons there can be at most n^2 different synapses of one type, each with its own set of equations. Thus, simulations can take very long times for large values of n. A solution to this problem is to implement some form of parallelisation in the ODE solver and the system of equations itself. One of the simplest methods of parallelizable computation is in the form of matrix algebra which can be accelerated using libraries such as BLAS which can only be used for accelerating CPU based computations. Similarly, CUDA is available for speeding up computations on Nvidia-based GPUs and TPUs. However, there is a barrier of entry to using low-level packages like CUDA for the general user as it sometimes requires an in-depth understanding of the architecture, particularly for troubleshooting.

This is where TensorFlow (an open-source Google product) gives us a massive edge. TensorFlow allows us much greater scalability and is way more flexible in terms of ease of implementation for specialised hardware. With minimal changes in the implementation, the code can be executed on a wide variety of heterogeneous and distributed systems ranging from mobile devices to clusters and specialised computing devices such as GPU and TPU cards. The modern advances in GPU/TPU design allow us to access even higher degrees of parallelisation. It is now even possible to have hundred of TeraFLOPS of computing power in a single small computing device. With TensorFlow, we can access these resources without even requiring an in-depth understanding of its architecture and technical knowledge of specific low-level packages like CUDA.

Materials and methods

Requirements for the Tutorial

For this tutorial, the reader is expected to have an understanding of Python and some commonly used packages such as Numpy and Matplotlib. The reader is also expected to know some amount of calculus particularly the theory of differential equations. Access to a computer will the following prerequisite software/packages is preferable: Python 3.6 or above, Jupyter Notebook, Numpy Python package, Matplotlib Python package, and TensorFlow 1.13 or above. All software can be installed using Anaconda Distribution of Python 3. Instructions for TensorFlow installation is available on their website.

Day 1: Of Numerical Integration, Python and Tensorflow

Our discussion begins with what Numerical Integration is and how we can use it to solve differential equations given the initial condition in Python using Numpy or TensorFlow.

What is Numerical Integration?

For a theoretician, the ideal form of the solution to a differential equation given the initial conditions, i.e. an initial value problem (IVP), would be a formula for the solution function. However, at times obtaining a formulaic solution is not easy, and in many cases it is impossible. So, what do we do when faced with a differential equation that we cannot solve? If one is only looking for long term behaviour of a solution, one can always sketch a direction field. This can be done without too much difficulty for some reasonably complex differential equations that we cannot solve to get exact solutions. However, what if we need to determine how a specific solution behaves, including some values that the solution will take? In that case, we have to rely on numerical methods for solving the IVP such as Euler's method or the Runge-Kutta Methods.

Euler's Method

We use Euler's Method to generate a numerical solution to an initial value problem of the form:

$$\frac{dx}{dt} = f(x,t) \tag{1}$$

$$x(t_o) = x_o \tag{2}$$

Firstly, we decide the interval over which we desire to find the solution, starting at the initial condition. We break this interval into small subdivisions of a fixed length ϵ. Then, using the initial condition as our starting point, we generate the rest of the solution by using the iterative formulas:

$$t_{n+1} = t_n + \epsilon \tag{3}$$

$$x_{n+1} = x_n + \epsilon f(x_n, t_n) \tag{4}$$

to find the coordinates of the points in our numerical solution. We end this process once we have reached the end of the desired interval.

Euler Method in Python

Let $\frac{dx}{dt} = f(x,t)$, we want to find $x(t)$ over $t \in [0,2)$, given that $x(0) = 1$ and $f(x,t) = 5x$. The exact solution of this equation would be $x(t) = e^{5t}$.

```python
import numpy as np
import matplotlib.pyplot as plt
def f(x,t): # define the function f(x,t)
    return 5*x
epsilon = 0.01 # define timestep
t = np.arange(0,2,epsilon) # define an array for t
x = np.zeros(t.shape) # define an array for x
x[0]= 1 # set initial condition
for i in range(1,t.shape[0]):
    x[i] = epsilon*f(x[i-1],t[i-1])+x[i-1] # Euler Integration Step
```

Vectorizing the Euler Method

Euler's Method also applies to vectors and can solve simultaneous differential equations. The Initial Value problem now becomes:

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}, t) \tag{5}$$

$$\vec{x}(t_o) = \vec{x_o} \tag{6}$$

where $\vec{x} = [x_1, x_2...]$ and $\vec{f}(\vec{x}, t) = [f_1(\vec{x}, t), f_2(\vec{x}, t)...]$.
The Euler's Method becomes:

$$t_{n+1} = t_n + \epsilon \tag{7}$$

$$\vec{x_{n+1}} = \vec{x_n} + \epsilon \vec{f}(\vec{x_n}, t_n) \tag{8}$$

Let $\frac{d\vec{x}}{dt} = f(\vec{x}, t)$, we want to find $\vec{x}(t)$ over $t \in [0, 2)$, given that $\vec{x} = [x, y]$, $\vec{x}(0) = [1, 0]$ and $f(\vec{x}, t) = [x - y, y - x]$.

```python
def f(x,t): # define the function f(x,t)
    x_,y_ = x
    return np.array([x_-y_,y_-x_])
t = np.arange(0,2,epsilon) # define an array for t
x = np.zeros((2,t.shape[0])) # define an array for x
x[:,0]= [1,0] # set initial condition
for i in range(1,t.shape[0]):
    x[:,i] = epsilon*f(x[:,i-1],t[i-1])+x[:,i-1] # Euler Integration Step
```

A Generalized function for Euler Integration

Now, we create a generalized function that takes in 3 inputs ie. the function $f(\vec{y}, t)$ when $\frac{d\vec{y}}{dt} = f(\vec{y}, t)$, the time array, and initial vector $\vec{y_0}$. The Algorithm for the Generalized Function is:

- Get the required inputs: function $\vec{f}(\vec{x}, t)$, initial condition vector $\vec{y_0}$ and time series t.
- Create a zero matrix to hold the output.
- For each timestep, perform the euler method updation with variable ϵ and store it in the output matrix.
- Return the output timeseries matrix.

```python
class _Integrator():
    def integrate(self,func,y0,t):
        time_delta_grid = t[1:] - t[:-1]
        y = np.zeros((y0.shape[0],t.shape[0]))
        y[:,0] = y0
        for i in range(time_delta_grid.shape[0]):
            y[:,i+1]= time_delta_grid[i]*func(y[:,i],t[i])+y[:,i]
        return y
```

```
def odeint_euler(func,y0,t):
    y0 = np.array(y0)
    t = np.array(t)
    return _Integrator().integrate(func,y0,t)
solution = odeint_euler(f,[1.,0.],t)
```

An Introduction to TensorFlow

TensorFlow is an open-source software library. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organisation to conduct machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well!

Essentially, TensorFlow library for high-performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers. It is a python package that (much like BLAS on Intel MKL) speeds up Linear Algebra Computation. What is unique about this system is that it is capable of utilising GPUs and TPUs for computation and its written in a more straightforward language like python.

Why GPU/TPU vs CPU?

The answer lies in the architecture: CPU = Faster per Core Processing, Slow but Large Memory Buffer, Few Cores GPU/TPU = Slower Processing, Faster but Smaller Memory Buffer, Many Cores

Thus, GPUs and TPUs have been optimised for a large number of simple calculations done parallel. The extent of this parallelisation makes it suitable for vector/tensor manipulation.

Euler Integration Function in TensorFlow

Transitioning to TensorFlow is not a trivial process, but after some practice, it is as simple as using Numpy. Because of the way the TensorFlow architecture is designed, there are a few limitations to how one can do simpler operations/manipulation. However, it is easy to overcome using the correct function and code patterns which can be easily learnt.

```
# Firstly, import TensorFlow
import tensorflow as tf

class _Tf_Integrator():
    def integrate(self, func, y0, t):
        time_delta_grid = t[1:] - t[:-1]
        # tf.scan(fn,el,init) is an iterator over elems, it
        # applies fn recursively  on tensor init fn is function
        # takes in two inputs: accumulated fn and the value of
        # current iteration on el
        y = tf.scan(scan_func, (t[:-1], time_delta_grid),y0)
        return tf.concat([[y0], y], axis=0)

    # Create and return a stepper function fn
    def _make_scan_func(self, func):
        def scan_func(y, t_dt):
            t, dt = t_dt
            dy = dt*func(y,t)
            return y + dy
        return scan_func

def tf_odeint_euler(func, y0, t):
    # Convert input to TensorFlow Objects
    t = tf.convert_to_tensor(t, preferred_dtype=tf.float64, name='t')
    y0 = tf.convert_to_tensor(y0, name='y0')
    return _Tf_Integrator().integrate(func,y0,t)

# Define a function using Tensorflow math operations.
# This creates a computational graph.
def f(X,t):
    x = X[:-1]
    y = X[1:]
    out = tf.concat([x-y,y-x],0)
    return out
y0 = tf.constant([1,0], dtype=tf.float64)
epsilon = 0.01
t = np.arange(0,2,epsilon)
# Define the final value (output of scan) that we wish to compute
state = tf_odeint_euler(f,y0,t)
# Start a TF session and evaluate state
with tf.Session() as sess:
    state = sess.run(state)
```

Results

Nulla mi mi, venenatis sed ipsum varius, Table 1 volutpat euismod diam. Proin rutrum vel massa non gravida. Quisque tempor sem et dignissim rutrum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi at justo vitae nulla elementum commodo eu id massa. In vitae diam ac augue semper tincidunt eu ut eros. Fusce fringilla erat porttitor lectus cursus, vel sagittis arcu lobortis. Aliquam in enim semper, aliquam massa id, cursus neque. Praesent faucibus semper libero.

Table 1. Table caption Nulla mi mi, venenatis sed ipsum varius, volutpat euismod diam.

Heading1				**Heading2**			
cell1row1	cell2 row 1	cell3 row 1	cell4 row 1	cell5 row 1	cell6 row 1	cell7 row 1	cell8 row 1
cell1row2	cell2 row 2	cell3 row 2	cell4 row 2	cell5 row 2	cell6 row 2	cell7 row 2	cell8 row 2
cell1row3	cell2 row 3	cell3 row 3	cell4 row 3	cell5 row 3	cell6 row 3	cell7 row 3	cell8 row 3

Table notes Phasellus venenatis, tortor nec vestibulum mattis, massa tortor interdum felis, nec pellentesque metus tortor nec nisl. Ut ornare mauris tellus, vel dapibus arcu suscipit sed.

LOREM and IPSUM nunc blandit a tortor

3rd level heading

Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque. Quisque augue sem, tincidunt sit amet feugiat eget, ullamcorper sed velit. Sed non aliquet felis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris commodo justo ac dui pretium imperdiet. Sed suscipit iaculis mi at feugiat.

1. react
2. diffuse free particles
3. increment time by dt and go to 1

Sed ac quam id nisi malesuada congue

Nulla mi mi, venenatis sed ipsum varius, volutpat euismod diam. Proin rutrum vel massa non gravida. Quisque tempor sem et dignissim rutrum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi at justo vitae nulla elementum commodo eu id massa. In vitae diam ac augue semper tincidunt eu ut eros. Fusce fringilla erat porttitor lectus cursus, vel sagittis arcu lobortis. Aliquam in enim semper, aliquam massa id, cursus neque. Praesent faucibus semper libero.

- First bulleted item.
- Second bulleted item.
- Third bulleted item.

Discussion

Nulla mi mi, venenatis sed ipsum varius, Table 1 volutpat euismod diam. Proin rutrum vel massa non gravida. Quisque tempor sem et dignissim rutrum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi at justo vitae nulla elementum commodo eu id massa. In vitae diam ac augue semper tincidunt eu ut eros. Fusce fringilla erat porttitor lectus cursus, vel sagittis arcu lobortis. Aliquam in enim semper, aliquam massa id, cursus neque. Praesent faucibus semper libero [3].

Conclusion

CO_2 Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque. Quisque augue sem, tincidunt sit amet feugiat eget, ullamcorper sed velit.

Sed non aliquet felis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris commodo justo ac dui pretium imperdiet. Sed suscipit iaculis mi at feugiat. Ut neque ipsum, luctus id lacus ut, laoreet scelerisque urna. Phasellus venenatis, tortor nec vestibulum mattis, massa tortor interdum felis, nec pellentesque metus tortor nec nisl. Ut ornare mauris tellus, vel dapibus arcu suscipit sed. Nam condimentum sem eget mollis euismod. Nullam dui urna, gravida venenatis dui et, tincidunt sodales ex. Nunc est dui, sodales sed mauris nec, auctor sagittis leo. Aliquam tincidunt, ex in facilisis elementum, libero lectus luctus est, non vulputate nisl augue at dolor. For more information, see S1 Appendix.

Supporting information

S1 Fig. Bold the title sentence. Add descriptive text after the title of the item (optional).

S2 Fig. Lorem ipsum. Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque.

S1 File. Lorem ipsum. Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque.

S1 Video. Lorem ipsum. Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque.

S1 Appendix. Lorem ipsum. Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque.

S1 Table. Lorem ipsum. Maecenas convallis mauris sit amet sem ultrices gravida. Etiam eget sapien nibh. Sed ac ipsum eget enim egestas ullamcorper nec euismod ligula. Curabitur fringilla pulvinar lectus consectetur pellentesque.

Acknowledgments

Cras egestas velit mauris, eu mollis turpis pellentesque sit amet. Interdum et malesuada fames ac ante ipsum primis in faucibus. Nam id pretium nisi. Sed ac quam id nisi malesuada congue. Sed interdum aliquet augue, at pellentesque quam rhoncus vitae.

References

1. Conant GC, Wolfe KH. Turning a hobby into a job: how duplicated genes find new functions. Nat Rev Genet. 2008 Dec;9(12):938–950.
2. Ohno S. Evolution by gene duplication. London: George Alien & Unwin Ltd. Berlin, Heidelberg and New York: Springer-Verlag.; 1970.
3. Magwire MM, Bayer F, Webster CL, Cao C, Jiggins FM. Successive increases in the resistance of Drosophila to viral infection through a transposon insertion followed by a Duplication. PLoS Genet. 2011 Oct;7(10):e1002337.