# Deep Learning from Ground Up

## Boot Up

Akshay Badola
15MCPC15

School of Computer and Information Sciences
University of Hyderabad

December 2017

## Overview

## Introduction

### What's Deep Learning?

- In the last 5 years or so Neural Networks have taken the world by a storm.
- Remarkable gains have been achieved by these so called **Deep Neural Networks**.

  For example:
  - Google used it to revolutionize its translate engine, professing that, "The A.I. system had demonstrated overnight improvements roughly equal to the total gains the old one had accrued over its entire lifetime."[1]
  - Networks claim to reach near human level performance on the task of image classification. [2]
  - In the challenging game of Go, computers beat human champions for the first time in 2014 [3][2]

[1] *The Great AI Awakening*. https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html. Accessed: 2017-12-01.

[2] For a detailed exposition on the pre AlphaGo attempts see: http://www.wired.com/2014/05/the-world-of-computer-go/

## What we're here for

Not a history lesson!

We're going to attempt to:

- Learn what and how these networks do from a ~~(somewhat) theoretical and (mostly) practical standpoint.~~
- Learn the building blocks of these nets and how we get to a simple neural network from a statistical viewpoint.
- Understand basic mathematical concepts required.
- Learn the programming tools and frameworks used for all this.
- Learn what are GPUs and why such accelerators are essential for this.
- Build and understand some of these networks by the end.

## What we're here for

Not a history lesson!

We're going to attempt to:

- Learn what and how these networks do from a (we'll see how much) theoretical and practical standpoint.
- Learn the building blocks of these nets and how we get to a simple neural network from a statistical viewpoint.
- Understand basic mathematical concepts required.
- Learn the programming tools and frameworks used for all this.
- Learn what are GPUs and why such accelerators are essential for this.
- ~~Build~~ and understand some of these networks by the end.

## Python

### Why Python

- Python is easy!

-
  ```
  print("Hello World")            # Very simple to print. No hassle!
  ```
  No semicolons, no type specifications. No memory management. Everything's handled internally.
- Python can be fast by using existing libraries.
  (C library) $->$ (Python Wrapper) $->$ (Call from Python). Simple!

## Python

### Why Python

- Python is easy!

-
  ```python
  print("Hello World")            # Very simple to print. No hassle!
  ```
  No semicolons, no type specifications. No memory management. Everything's handled internally.
- Python can be fast by using existing libraries.
  (C library) $->$ (Python Wrapper) $->$ (Call from Python). Simple! Simple to use but not to build ourselves, though some tools exist to help. [4, 1]
- Python is powerful as it's a general purpose programming language and a specialized tool for someone who doesn't want to interface with the system.
  - You can create threads and call system processes.
  - File reading and writing is a snap!
  - You can interface with the system libraries to leverage other tools to assist you.
  - You can create automated tools all in python which can integrate easily with your code and help you monitor your algorithms, test and debug.

## Python contd.

### Why Python

- Python is a scripted language so it's easy to run and evaluate statements. No long compile times!
- Python has what's called *duck typing* and garbage collection via reference counting.
- Python has developed a HUGE ecosystem. There are tonnes of libraries!
- Python numpy interfaces with the BLAS and LAPACK libraries if they exist on your system so linear algebra is really fast.
- There's lots of example code out there for you if you get stuck, including Neural Networks, Statistical Models, Simulations, Machine Learning Models etc.
- We'll use *iPython*.

## Python contd.

### Intro to Python

- These are equivalent

```python
print('Hello world')
print("Hello world")
```

- Functions are simple

```python
def add(num1, num2):
    num3 = num1 + num2
    return num1, num2, num3
```

- A lot of the syntax is C style

```python
if foo == "var":
    func()
else:
    pass
```

## Python contd.

### Intro to Python

- A lot of the syntax is C style

```
for i in range(100):
    print(i)

j = 10
while j:
    print("looping")
    j -= 1

# There's no ++ or -- though. It just avoids confusion.
# For more details see http://csapp.cs.cmu.edu/3e/docs/chistory.html
# Nor is there a switch statement. Use multiple if/else clauses
```

- There are a few basic data types, but each type is wrapped
  *Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute doc, which returns the doc string defined in the function's source code. The sys module is an object which has (among other things) an attribute called path. And so forth.*

- `print(int.__doc__, sum.__doc__) ## Here int is a type and sum is a function`

## Python contd.

### Operators and Strings

- With standard C style operators:

```
3 + 2                               # addition
3 - 2                               # subtraction
3 * 2                               # multiplication
3 % 2                               # remainder
(3 + 2) ** 3                        # exponentiation
```

## Python contd.

### Operators and Strings

- With standard C style operators:

```
3 + 2                                # addition
3 - 2                                # subtraction
3 * 2                                # multiplication
3 % 2                                # remainder
(3 + 2) ** 3                         # exponentiation
```

- Strings can be specified with either a single or double quote

```
'spam eggs'          # single quotes
'doesn\'t'           # use \' to escape the single quote...
"doesn't"            # ...or use double quotes instead
'"Yes," he said.'    # same thing with single and double quotes inverted
"\"Yes,\" he said."
'"Isn\'t," she said.'                # a mix
'C:\some\name'                       # What does this print?
```

## Python contd.

### Data Structures

- Multiline strings

```
r'C:\some\name'  # note the r before the quote
# A string literal starting and ending with """ can span multiple lines
"""
Something
    Something else
Something else else
"""
# They're quite simple
'1' + '2'                           # Can be easily concatenated
'1' - '2'                           # Though not subtracted
'1'*3                               # Can be multiplied
'1'/3                               # but not divided
```

## Python contd.

### Data Structures

- Multiline strings

```
r'C:\some\name'  # note the r before the quote
# A string literal starting and ending with """ can span multiple lines
"""
Something
    Something else
Something else else
"""
# They're quite simple
'1' + '2'                       # Can be easily concatenated
'1' - '2'                       # Though not subtracted
'1'*3                           # Can be multiplied
'1'/3                           # but not divided
```

- The simplest data structure. They're a bit like the strings.

```
def func():
    pass                                   # does nothing
[1, 2, 3, "somestring", [func, [1, 2, {}]]]  # A list can hold anything
[]                                          # Empty list
[1, 2, 3] + [func, [1, 2, {}]]  # Gives [1, 2, 3, func, [1, 2, {}]]
[1, 2, 3].append(23)            # cannot do [1, 2, 3] + 23
```

## Python contd.

### Data Structures

- But the two can't be mixed.

```
[1] + 'string'                    # Gives error
print([1], 'string')              # print converts it automatically though
```

- In fact, in most cases you can't mix types. There's no implicit type conversion. But if types can mix, they're usually subtypes of a more general type. Like both Integer and Float are subtypes of Number, and Number has certain properties which allows them to be added/subtracted etc. together.

```
2 + 3.5                             # int + float

for i in [1, 2, 3]:                 # iterating over a list
    print(i)

for i in range(10):                 # Careful, range is not a list!
    print(i)

[1, 2, 3] + range(10)           # Doesn't work
range(10)[0]                    # range type can also be subscripted. Why?
```

## Python contd.

### Data Structures

- Although explicit type conversions are allowed.

```
# But the types have to support that
str(2) + '345or_some_text'        # works
[1, 2, 3] + list(range(10))       # also works
```

- List indexing:

```
l = list(range(100))              # lists are also zero indexed
l[0: 20]                          # Start from 0, take 20 elements
l[0: 20: 3]                       # 0 to 20, but step_size == 3
m = l                             # m is a reference to l
m[0] = 'test'                     # l also changes

# But if we allocate new memory for m
l = list(range(100))
m = l[0: 20: 3]
m[0] = "test"                     # This will not change l

# However, with types that are not dynamic
u = 3; v = u                      # (not recommended style)
v = 2; print(u)                   # u doesn't change
```

## Python contd.

### Exercises!

- What will the following code print and why?

```
print('test'[0])
print([1, 2, 3] * 3)
l = [[1, 2, 3] * 3]
l[0][0] = 0; print(l)
l = [[1, 2, 3]] * 3
l[0][0] = 0; print(l)
```

- Scope:

```
a = 10
def func():
    a = 5
    print(a)                        # prints 5
print(a)

def func():
    print(a)                        # prints 10
```

## Python contd.

### Scope

- Scope contd.:

```
def func():
    a = 5
    for i in range(1):
        a = 2
    print(a)                         # prints 2. for has modified a as a was in sco
func()
print(a)                             # prints 10 as the variable in global scope is

a = True; b = False
if a:
    c = True
print(c)                             # c exists, is not local to if
del a, b, c

a = True; b = False
if b:
    c = True
print(c)                             # c doesn't exist
```

## Python contd.

### List Operations

Lists are objects of type `iterator`. (Numpy) Arrays can also be iterated.

- You should lookup these common functions on lists yourself: index, pop, append, copy, reverse, sort
- Some more examples are given at this link `http://openbookproject.net/thinkcs/python/english3e/lists.html`
- We'll demonstrate some list comprehensions and move on to dictionaries, sets and numpy.

```
a = [l for l in range(10)]       # equivalent to a = list(range(10))
a = [l for l in range(10) if l % 2 == 0]  # list with a condition
a = [l ** 2 for l in range(10)
     if l % 2 == 0]            # list with a condition and an expression
a = [func(str(l), l ** 2)
     for l in range(100)
     if l  % 2 == 0]             # Even more complicated

# Also notice that the following isn't allowed
a = func(str(b), b) /
    2
a = func(str(b), b) / \
    2                            # But this is
```

## Python contd.

### Dictionaries and sets

Dictionaries are unordered (key, value) pairs.

```
a = {}                              # empty dictionary
a = {'one': 1, 2: 2, 'three': [1, 2, 3],
     'four': {'one': 1, 'two': 2}}  # keys can be anything
a['one']                            # retuns 1
a = {['one', 'two']: 1, 'two': 2}  # But not types that can change
b = set()                          # empty set
b = {'one', 2, 'three'}

'one' in a                         # membership test, checks only in keys
'two' in b; 1 in [1, 'two', 3]     # works for lists too

for k, v in a.items():             # a.iteritems() for python2
    a[k] = func(v)

a.keys(); a.values()               # To get only keys or values
```

## Python contd.

### Dictionaries and Sets compared to Lists

Dictionaries are much faster because they're indexed.

```
# simple operaton for testing existence
5 in [1, 2, 3, 4, 5]               # Returns True
5 in {1, 2, 3, 4, 5}               # Returns True
2 not in [1, 2, 3]                 # Returns False

# The latter should be faster, but internal optmizations render it the same
%timeit 5 in [1, 2, 3, 4, 5]
%timeit 5 in {1, 2, 3, 4, 5

[1, 2, 3, 4, 5].index(5)           # Returns 4
{1, 2, 3, 4, 5}.index(5)           # Sets don't have an index

a = [1, 2, 3, 4, 5]; a.pop(4)      # Returns and deletes 5 from list
a = {1, 2, 3, 4, 5}; a.pop(4)     # Error. set.pop() pops random item
a = [1, 2, 3, 4, 5]; a.remove(4)     # Removes 4 from the set
```

## Python contd.

Very quickly classes!

```
# some more function definition and usage
def foo(a, b=10, *args, **kwargs):
    print(a)
    print(b)
    for arg in args:
        print(args)                # simply print
    for k, v in kwargs:
        print(kwargs)

foo(a)                             # b defaults to 10, args, kwargs are optional
foo(a, 5, [1, 2, 3], {'one': 1, 'two': 2})  # passing args and kwargs


class Foo():                # can be anything but capitalized by convention
    a = 10                  # class member. Common to all instances
    def __init__(self, arg1, arg2='test', *args, **kwargs):  # constructor/initial
        self.var = arg1  # accessible to all functions of class but not outside

    def foo(self, args):           # Note the 'self' keyword here and in __init__
        pass

foo = Foo(a, b)                    # *args and **kwargs are optional
```

## Python contd.

### Instantiating and Communicating among Classes

```
class Foo():                    # equivalent to class Foo(object)
    def __init__(cls, arg, *args, **kargs):
        cls.arg = arg
        cls.__arg = 'hidden'

    def func(cls, arg):
        cls.arg = arg
        print(cls.arg)

foo = Foo('test')
print(foo.arg)                        # prints test
foo2 = Foo('test2')
print(foo2.arg)                       # prints test2
foo.__arg                             # no such variable? why?
foo._Foo__arg                         # additional measure to hide
```

## Python contd.

### Instantiating and Communicating among Classes

```
class Foo():                  # equivalent to class Foo(object)
    def __init__(self, cls, arg, *args, **kargs):
        self.arg = arg
        self.cls = cls

    def func(self):
        print(self.arg)
        self.cls.func_2()

class Foo_2():
    def __init__(self, arg):
        self.arg = arg

    def func_2(self, arg=None):
        if arg:
            print(arg)
        else:
            print(self.arg)

foo2 = Foo_2('string')
foo = Foo(foo2, 'test')
foo.func()         # the function also calls the other object's function
```

## Python contd.

### Modules, OS and file handling

```
import os
import sys, io
from os import path
import os.path as path          # both do the same thing

# but if only the first style is used then you have to specify the full module path
os.path.exists('/home/user')   # checks if the file path exists
os.path.join('/home', 'user')  # joins a path taking care of the / 'es
os.path.isdir('/home/user')    # os.path.isfile(), os.path.islink()
os.path.abspath('.')           # gives full path of the relative path
os.mkdir('/home/user/test_dir')  # makes a directory os.rmdir()

f = open('test', '2')                # opens the file for reading in the pwd
content = f.write('content')
f.flush()
f.close()
```

## Python contd.

### *with* syntax

- *with* syntax is used with objects that support some automatic exception handling

```
with open('test', 'rw') as f:
    old_content = f.read()
    f.write('some_new_content')
# closing the file is handled automatically
```

- The cleanup code like file closing, memory freeing (though not required because of python's GC), foreign/C objects freeing etc. is performed automatically, and if an error occurs, it is safely thrown.
- You can have your own function be compatible with the the *with* clause but we won't go into that. Search online for it!

## Python contd.

### Environment: anaconda, ipython and pip

Anaconda

- Download `https://www.anaconda.com/download/#linux`
- Install `https://conda.io/docs/user-guide/install/index.html`
- Set path before using.
  $>export PATH=`path_to_conda/bin`:$PATH.
- conda help
- Anaconda ships with a lot of pre-compiled packages so you don't have to maintain them.

## Python contd.

### pip

pip: Python package management

- apt-get install python3-pip. dnf install python3-pip. zypper install python3-pip
- pip3 search <package_name>. Searches packages at PyPI.
- pip3 install. Installs packages.
- pip3 freeze. Lists install packages.
- pip3 install -U. Forces an upgrade
- pip3 uninstall. Uninstalls packages
- pip3 install –user. Installs in the user namespace.
- As an exercise to get familiar with it, we'll install the packages we'll be using using pip
- pip3 install –user -U numpy scipy matplotlib ipython scikit-learn scikit-image pillow

## Python contd.

### pip

pip: Python package management

- apt-get install python3-pip. dnf install python3-pip. zypper install python3-pip
- pip3 search <package_name>. Searches packages at PyPI.
- pip3 install. Installs packages.
- pip3 freeze. Lists install packages.
- pip3 install -U. Forces an upgrade
- pip3 uninstall. Uninstalls packages
- pip3 install –user. Installs in the user namespace.
- As an exercise to get familiar with it, we'll install the packages we'll be using using pip
- pip3 install –user -U numpy scipy matplotlib ipython scikit-learn scikit-image pillow
- Because you won't have access to GPUs so won't really need anaconda. You can learn to use it as the need arises.

## Python contd.

### iPython!

We also have iPython!

- Start by opening up ipython. Type `ipython3` at the terminal.
- help(something) shows help for it.
- tab auto completes a variable (functions are also variables, sort of).
- `Ctrl-{a,e,b,f}` go `{start-of-line, end-of-line, back-one-char, forward-one-char}`
- `Alt-{b,f}` go `{back-one-word, forward-one-word}`
- `Ctrl-{p,n,k,y}` do `{previous-command, next-command, kill-line (cut), yank (paste, not necessarily a line)}`
- `Ctrl-{Space, w}` do `{set-mark, kill-region (can use C-y to yank)}`
- `Ctrl-{r,s}` do `{search-command-backward, search-command-forward}`
- `Alt-{d,\}` does `{delete-word, delete-horizontal-space}`

## Python contd.

### more iPython!

- `%quickref` lists a whole lot of useful commands. You can check it out at leisure.
- `%run` runs a python script from iPython.
- `%paste` pastes text preserving formatting, from clipboard (requires tk installed in ubuntu).
- `%timeit`, `%%timeit` time a single line and multiline commands.
- `%pdb` turns ON/OFF automatic python debugger.
- `lsmagic` lists all the available iPython macros (magics).
- If you have a newer version of iPython, `automagic` will be on so you can just do `run ./myscript.py` instead of `%run ./myscript.py`.

## Some String Operations

- String is not really a list in python but it feels like it.

```
list('abcde')
Out[312]: ['a', 'b', 'c', 'd', 'e']  # returns a list

# str has some simple functions for convenience
a = 'This is a test'
a.startswith('This')              # True
a.endswith('test')                # True
'donkey' in a                     # False
'test' in a                       # True
a.find('test')                    # returns the index 10
a.upper()                         # returns upcase string. DOESN'T CONVERT STRIN
a.lower()
a.strip()                         # Removes surrounding whitespace
a.split()                         # returns a list of words, breaking the string
a.split('t')                      # now breaks at letter 't', is useful for remo
a.replace('This', 'Miss')         # Replaces the said string

# We'll briefly mention map, filter
map(int, ['1', '2', '3', '4', '5'])  # Converts to a list of numbers
filter(lambda x: x < 3, [1, 2, 3, 4, 5])  # Filters elements satisfying condit

# lambda is a simple anonymous function, for quick tasks!
lambda x: True if x[0] else False  # What does this do?
lambda *args: print([type(arg) for arg in args])  # What does this do?
f = lambda x: True if x[0] else False  # Not recommended syntax. Use def
```

## Python contd.

### Exercises! (again)

- Write a small lambda function that given a number (say 4) generates a list of tuples like so

-
  ```
  [(1, 2), (2, 3), (3, 4), (4, 5)]
  ```
- Now instead of the second number simply being one greater than the previous one, have it be the output of a user supplied function.

  ```
  func(4, ufunc)
  ```

  gives

  ```
  [(1, ufunc(1)), (2, ufunc(2)), (3, ufunc(3)), (4, ufunc(4))]
  ```
- How do you sort a list of tuples according to just the first number of each tuple? [hint: look at the documentation of list.sort]
- Now, what's the simplest (perhaps a pythonic) way to convert this list (or any list) of tuples to a dictionary?

## Python contd.

### Exercises! (cond.)

-
  ```
  # argparse is a module that parses command line arguments in python
  import argparse

  def main():
      parser = argparse.ArgumentParser(description="test")
      parser.add_argument('--datadir', '-d', type=str, default='.',
                          help='location of the log data')
      args = parser.parseargs()
      print(args.datadir)

  main()
  ```
- Let's say I wanted to give the arguments parser a list to parse. Notice the `type=str`. That determines the type of the argument which is parsed. But there's no inbuilt datatype list that it can parse. How can you parse it?
- Create a class that simply keeps track of the number of instances it has, and prints the total number each time a new instance is created. [hint: recall the difference between class variables and instance variables]

## Python contd.

### Exercises! (contd.)

- Just like `map`, `reduce` is a function that takes a list but it also takes an intial value as an argument, and instead of simply applying the function on each element of the list, it applies it along with the initial value and continues it through the list until the list is reduced to a single value.
  Can you use it to `join` a list of lists into a single list?

```
from functools import reduce
# help(reduce) gives reduce(func, sequence[, initial]) -> value
# turn this [[1,2,3], [3,4,5], [4,5,6]] to [1,2,3,3,4,5,4,5,6]
```

- Write a complete program which takes a directory as argument, reads all the files in there with a '.py' extension and for each file, lists the following to stdout.
  1. All the import modules with a fully qualified name, .e.g., `from os import path` should be shown as `os.path`
  2. All the classes in the files, with all the functions
  3. All the functions which are not members of any class.

## Python (contd.)

### Exercises! (contd.)

- Write a program that reads a CSV file which has ; as the delimiter,  as comment and it takes the last two columns, lowercases the strings and writes it to another CSV file.

## Discussion

- What do you think is Deep Learning? What is Machine Learning first of all?

## Discussion

- What do you think is Deep Learning? What is Machine Learning first of all?
- How and what can a machine learn?

## Discussion

- What do you think is Deep Learning? What is Machine Learning first of all?
- How and what can a machine learn?
- Can Machines be Creative?

## Discussion

- What do you think is Deep Learning? What is Machine Learning first of all?
- How and what can a machine learn?
- Can Machines be Creative?
- Can a machine become sentient? What is sentience anyway?

## Discussion

- What do you think is Deep Learning? What is Machine Learning first of all?
- How and what can a machine learn?
- Can Machines be Creative?
- Can a machine become sentient? What is sentience anyway?

## References I

[1] *Automatic Wrapper and Interface Generator*. https://github.com/StatisKit/AutoWIG. Accessed: 2017-12-01.

[2] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[3] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[4] *Simplified Wrapper and Interface Generator*. http://www.swig.org/. Accessed: 2017-12-01.

[5] *The Great AI Awakening*. https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html. Accessed: 2017-12-01.