BUSINESS ANALYTICS CLUB

Workshop Series 9.26

Machine Learning

Janet Ye
Shantanu Joshi

BAC

Set Ups

- Download the zip file from bit.ly/bacdata "Python ML" folder. In the zip file, you'll find:
  - Installation Guide
  - Python Installers
  - Sublime Text (Text editor) Installers
  - Windows Users:
    - Special Library Package File (IMPT: DO NOT edit the name of this file)

BAC

Learning Objective

1. Understand the fundamentals of a Neural Network and it's application through Python
2. Install and use basics of Python 2.7
3. Setup NumPy
4. Introduce Sigmoid Curves and Backpropagation

Terminology

Machine Learning: the study and construction of algorithms that can learn from and make predictions on data

Neural Network: statistical learning method inspired by biological neural networks

Backpropagation: how our Neural Network will learn…

Sigmoid Function: can map any number to be between 0,1

BAC

NumPy Setup

Numpy on Mac

- NumPy is a library in python that adds support for arrays or data frames
- Mac Users: Type [Command+Space] and type [Terminal], hit enter to open a new terminal window
  - In the terminal window type [pip install numpy] and hit enter

Numpy on Windows

- Installation is tricky. Follow instructions carefully.
- Type [Windows+Q] and type [powershell], double click to open [Windows PowerShell]
- In PowerShell, type:

  cd .\Downloads\python_mlworkshop\Windows\

  hit ENTER, then type:

  pip install numpy

  and instead of hitting ENTER this time, hit the [TAB] key, this should autocomplete to:

  numpy-1.9.3+mkl-cp27-none-win32.whl

  now hit ENTER
- Note: PowerShell will be referred to as the Terminal in the rest of the presentation.

Testing Numpy

- Open a new Terminal window (Windows: PowerShell)
- Type: python
- Once the Python prompt appears type: import numpy as np and hit ENTER, nothing should happen
- Type: np and hit ENTER the path of numpy should appear

Building our File

- Open Sublime Text and create a new file called neural.py
- Save to your DESKTOP
- The bottom right corner of sublime text should say [Python]
- Type: print "Hello World" and hit save
- Open Terminal, type cd Desktop. You should see:
  Mac: Desktop your_username$
  Windows: C:\Users\your_username

BAC

Running a Python File

Once you are in Desktop directory…

- Typing [ls] on mac or [dir] in windows will display [neural.py]

Running a file:

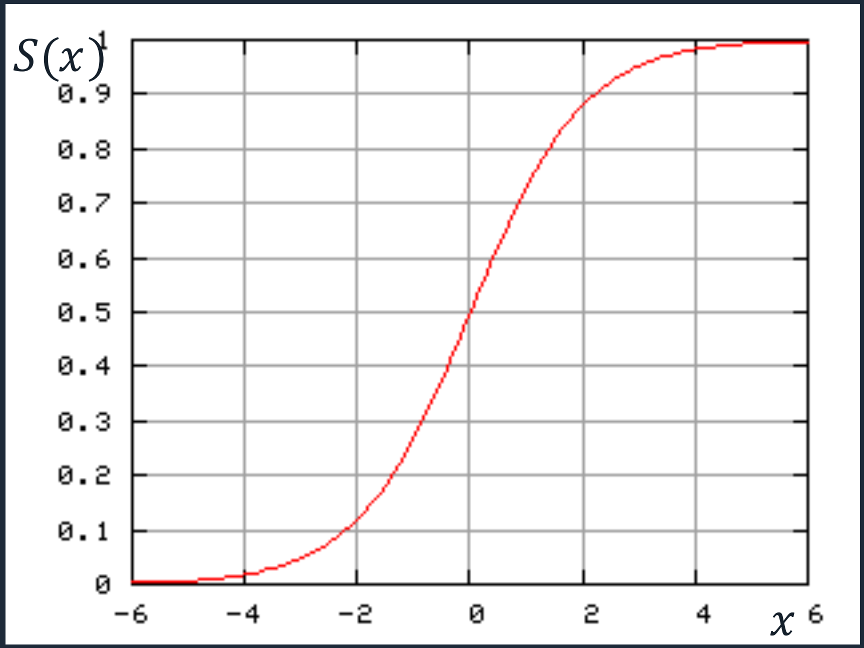
- Type: python neural.py
- You should see Hello World printed on your screen

BAC

Sigmoid Function

Sigmoid Function

- Maps any value to a value between 0 or 1
- Can convert numbers to probabilities
- $\frac{dS}{dx} = x * (1 - x)$



$$S(x) = \frac{1}{1 + e^{-x}}$$

BAC

Importing Numpy

In Sublime, edit neural.py delete the previous text and leave only:

```
import numpy as np
```

Syntax:

- Importing NumPy in this way allows us to use all the prebuilt functions in the NumPy package
- Setting numpy as np simply allows us to use np instead of typing out numpy

Input the Sigmoid Curve

In Sublime, edit neural.py and type →

```
# sigmoid function
def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))
```

Syntax:

- def: tells python we are defining a function called nonlin with parameters x and deriv
- deriv is an optional variable, and it is set to False by default. You will see in a few slides how to change its value.

BAC

Python Indentation

VERY IMPORTANT!

- Indents in python are considered syntax.
- 1 indent = 4 spaces
- Nested items require an extra indent than their parent items.
- Incorrectly placed or missing indents lead to most python syntax compline errors.

Input the Sigmoid Curve

- return tells our function to output something <u>and</u> terminate the function
- The next line is an if-statement, if deriv is set to True then the nested statement is returned
- If deriv is False, it skips the if statement and the last line is returned

```python
# sigmoid function
def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))
```

BAC

Note: NumPy Usage

- Notice the np.exp(-x)
- As previously defined we set numpy as np so this function actually reads as:
  numpy.exp(something)

```python
# sigmoid function
def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))
```

- The numpy.exp function is a function that returns the value of [e] (~2.718…) raised to the [something] power

BAC

Sigmoid Defined

- In the function nonlin we've defined the behavior of our sigmoid curve
- When [deriv=false] the function tells us $\frac{1}{1+e^{-x}}$
- When [deriv=true] the funciton outputs the derivative at a given [x] which is: $1 * (1 - x)$

BAC

Initializing the Dataset

Our Data Set

Inputs: (1)	(2)	(3)	Output
0	0	1	0
1	1	1	1
1	0	1	1
0	1	1	0

- We are trying to predict the Output column via the three Input columns
- Measuring statistics ex) Leftmost column is perfectly correlated with output will help solve the problem

Initializing the Inputs

- Type the following after the nonlin definition:

```
# input dataset
X = np.array([  [0,0,1],
                [0,1,1],
                [1,0,1],
                [1,1,1] ])
```

Syntax:

- We are setting X to be a matrix representing our inputs
- np.array is the NumPy function for multidimensional arrays
- Can also be written in one line (no need of special spacing)

    X = np.array([[0,0,1], [0,1,1], [1,0,1], [1,1,1]])

Initializing the Output

- Type the following after the input definition:

```
y = np.array([[0,0,1,1]]).T
```

Syntax:

- We are setting y (note: lowercase) to be a vector representing our output
- The .T at the end of the array definition is the transpose function in numpy
- Why? Because It's easier to write .T then np.array([[0],[0],[1],[1]])

BAC

Array Transpose…

$$y = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$$

$$y^T = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

BAC

Initializing Random Seed

- Type the following after the output definition:

```
np.random.seed(1)
```

Syntax:

- This refers to the seed number for the random function in the np.random module
- Random numbers come from normal distributions, the seed allows each random number to come from the SAME normal distribution
- That way we train the model using the same random numbers. This allows us to see the true effects of our neural network.

Initializing Weights

Initializing Weights

- Type the following after the random seed function:

```
# initialize weights randomly with mean 0
syn0 = 2*np.random.random((3,1)) - 1
```

Syntax:

- Initial weights will be set to syn0, the first layer synapse
- np.random.random((r,c)) returns an r by c matrix (in this case 3x1) of random numbers in the range [0, 1), based off of the seed
- 2 * np.random.random – 1 manipulates the interval and returns random numbers in the rage [-1, 1)

BAC

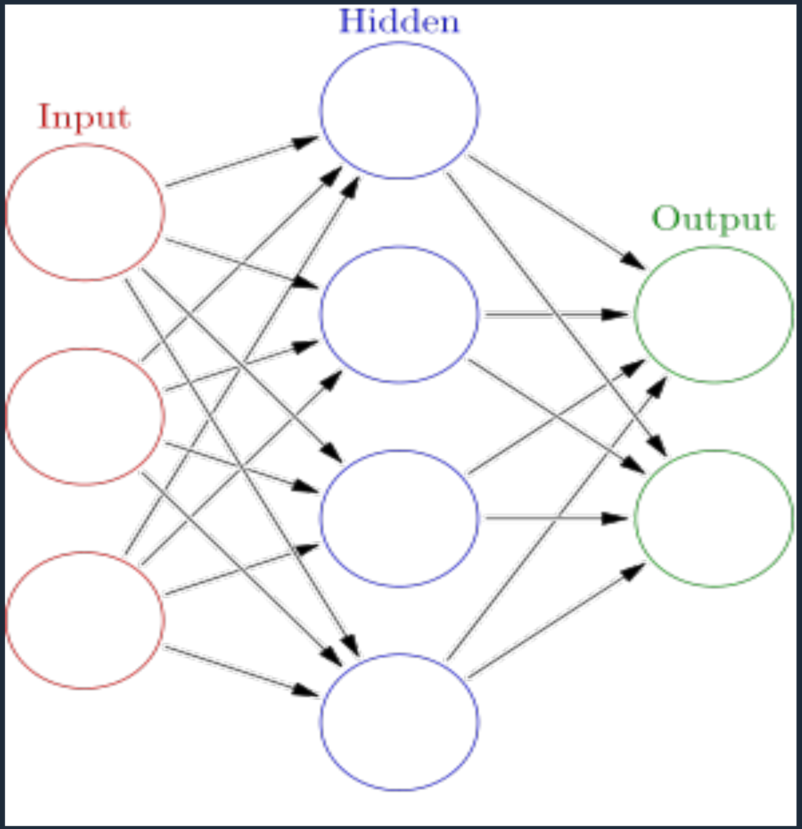
Understand Neural Networks

Human Neurons

BAC

Neural Network

- Neural network is a machine learning algorithm that tries to mimic the human brain.
- Widely used in the 80s and early 90s, but popularity diminished because it requires intense computational power.
- Recent resurgence: state-of-the-art technique for many applications



BAC

Training Session

```
for iter in xrange(10000):

    # forward propagation
    l0 = X
    l1 = nonlin(np.dot(l0,syn0))

    # how much did we miss?
    l1_error = y - l1

    # multiply how much we missed by the
    # slope of the sigmoid at the values in l1
    l1_delta = l1_error * nonlin(l1,deriv=True)

    # update weights
    syn0 += np.dot(l0.T,l1_delta)
```

Breaking it down line by line…

Training Session

- We begin a loop by writing: `for iter in xrange(10000):`
- (Don't miss the colon at the end!)
- xrange (10000) means we are looping iter, our placeholder variable, from 0 to 9999. You can name it anything you'd like, usually people use something like i, or iter
- The loop first sets iter to 0, computes everything that is indented under the for loop.
- Then set iter to 1, compute…set to 2, compute…etc. The loop stops after iter completes the 10000th round. The value of iter at completion is 9999.

BAC

Training Session

- Forward propagation: going forward with the layers
- Sublime text should automatically begin indented block within the for loop
- `l0 = X` here we set the input layer of the neural network to X
- `l1 = nonlin(np.dot(l0,syn0))` computes the hidden layer.
  - np.dot(l0, syn0) is the dot product calculation for matrices
  - Then we pass the dot product to the nonlin function we wrote
- How did we do? `l1_error = y - l1` compares our prediction (what sigmoid function outputs) with the actual y

BAC

Training Session

```
l1_delta = l1_error * nonlin(l1,deriv=True)
```

- Recall in Calc 1, Euler's method says dy = dx * slope
- l1_error is like your dx
- nonlin(l1, True) calls the nonlin function. It sees the deriv=True, and enters into the if statement

```
# sigmoid function
def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))
```

- l1_delta is like your dy

BAC

Training Session

```
syn0 += np.dot(l0.T,l1_delta)
```

- Computer science syntax: i += 1 is the shorthand for i = i + 1 (Similarly, i -= 1 is the same as i = i – 1)
- We now adjust syn0, the first layer synapse, by adding an adjustment term – the dot product of l0 and l1_delta
- This step is called backward propogation
- The loop continues…iter is incremented by 1

- So far, you should have something like this.
- Take a second to modify your code if need be.
- Pay attention to indentation, commas, and colons

```python
import numpy as np

# sigmoid function
def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))

# input dataset
X = np.array([  [0,0,1],
                [0,1,1],
                [1,0,1],
                [1,1,1] ])

# output dataset
y = np.array([[0,0,1,1]]).T

# seed random numbers to make calculation
# deterministic (just a good practice)
np.random.seed(1)

# initialize weights randomly with mean 0
syn0 = 2*np.random.random((3,1)) - 1

for iter in xrange(10000):

    # forward propagation
    l0 = X
    l1 = nonlin(np.dot(l0,syn0))

    # how much did we miss?
    l1_error = y - l1

    # multiply how much we missed by the
    # slope of the sigmoid at the values in l1
    l1_delta = l1_error * nonlin(l1,deriv=True)

    # update weights
    syn0 += np.dot(l0.T,l1_delta)

print "Output After Training:"
print l1
```

Running Code

- In Terminal, (Desktop should be your current directory), type python neural.py
- You should get something like…results are quite good

```
MacBook:ml shantanu$ python neural.py
Output After Training:
[[ 0.00966449]
 [ 0.00786506]
 [ 0.99358898]
 [ 0.99211957]]
```

Real Life Example: Self-Driving Car

Useful Resources

- Free Stanford Machine Learning on Coursera
  - Link: https://www.coursera.org/learn/machine-learning
  - Blog with detailed write-ups http://www.holehouse.org/mlclass/index.html
- Reddit: https://www.reddit.com/r/MachineLearning/wiki/index
- Machine Learning Data Sets: http://archive.ics.uci.edu/ml/
- Machine Learning Visualization: http://www.r2d3.us/visual-intro-to-machine-learning-part-1/
- Dope Blog: http://www.wzchen.com

Acknowledgement

Andrew Trask

Product Manager @
Digital Reasoning

BAC

Appendix – Command Line Navigation

Navigation Tips:

- Type [cd] to change directories
- [cd ..] to move up a directory
- Type [ls] on mac or [dir] in windows to see contents of your current directory
- Typing tab will autocomplete with items from your current directory
- Typing [pwd] on a mac or [cd] in windows will list the current directory path