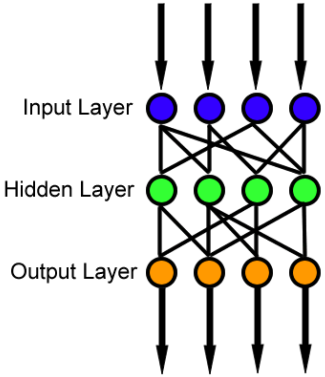
Introduction to Python



+ Artificial Neural Networks

INTRODUCTION

What is Python?

- Python is a high-level, multi-purpose, object oriented programming language
- It is run on an interpreter and has cross platform capability Applications of Python include:
  - Web development
  - Scientific computing, modelling, analysis
  - GUI development
  - Android development

Where do I get it?

- To install Python on your device, you can go to https://www.python.org/downloads/ and download the latest version of Python (Python 3.4.3)
- Or, you can run your code on the fly with a web interpreter http://www.tutorialspoint.com/python/index.htm (note you cannot install additional libraries or access local programming functionality of your computer if you use this)

- Python is an *object oriented* language that allows for procedural programming and execution, and allows for convenient manipulation of data structures

- Python has built-in *functional programing* characteristics (similar to R and Matlab) that make manipulation of lists and other data structures simple

In this lesson, we will do a crash course in Python, cover an application example and then get introduced to neural networks and machine learning.

PYTHON FUNDAMENTALS

1. Syntax
   - Python has no statement termination characters and functional blocks are indicated by indent. This indent is extremely important. Your code will not execute if it is not properly indented.

Example:

```
rangelist = range(10) # range = [1,2,3,…,9,10]
for number in rangelist: # for loop
        if number in (3, 4, 7, 9): # if then statement
                break # break out of loop
        else:
                continue # start next iteration of loop

if rangelist[1] == 2: # if else if  else block
        print "64s1c 6!tch35"
elif rangelist[1] == 3:
        print "The second item (lists are 0-based) is 3"
else:
        print "Yolo“

while rangelist[1] == 1:
        pass
```

http://www.stavros.io/tutorials/python/

Types and arithmetic

Variables of a certain *type* are implicit in Python. Python creates variables on the fly. However, variable *identifiers* (names) must be unique, not be Python keywords and are case sensitive.

Example:
To declare several numeric variables in Python:The assignment is done with the single equals '=' operator. A *list* is an updatable array of primitive data types. Data types do not have to be the same.

```
this_is_an_integer = 12345
a_long_val = -51924361L
float_val = 32.3+e18
complex_val = 5.3212 + 3.14j
```

Lists are addressed starting from zero with list[a:b]; *a* and *b* can be left out to specify the beginning or end of the list.

```
some_list = ["string_literal" , 12345, 12.5,  ["snakes in this nested list!" ] ]
some_list[3][0] = "no more snakes"
```

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

http://www.tutorialspoint.com/python/python_basic_operators.htm

Logical comparison operators. Be careful and compare same types only!

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

http://www.tutorialspoint.com/python/python_basic_operators.htm

Statement	Description
if statements	An if statement consists of a boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else ifstatement(s).

while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Functions in Python take arguments by reference and return a value or return nothing.

```
def functionname( parameters ): "function_docstring" function_suite

...

return [expression]
```

```
def sum( arg1, arg2 ): # Add both the parameters and return them."
                total = arg1 + arg2
                print "Inside the function : ", total
                return total;

# Now you can call sum function
total = sum( 10, 20 );
```

APPLICATION EXAMPLE

We can put everything together and develop a program of practical use. We will implement a simple Prime Number sieve in Python. We will utilize lists and functions. The algorithm is as follows:

1. Create a list of consecutive integers from 2 through *n*: (2, 3, 4, ..., *n*).

2. Initially, let *p* equal 2, the first prime number.

3. Starting from *p*, enumerate its multiples by counting to *n* in increments of *p*, and mark them in the list (these will be 2*p*, 3*p*, 4*p*, ... ; the *p* itself should not be marked).

4. Find the first number greater than *p* in the list that is not marked. If there was no such number, stop. Otherwise, let *p* now equal this new number (which is the next prime), and repeat from step 3.

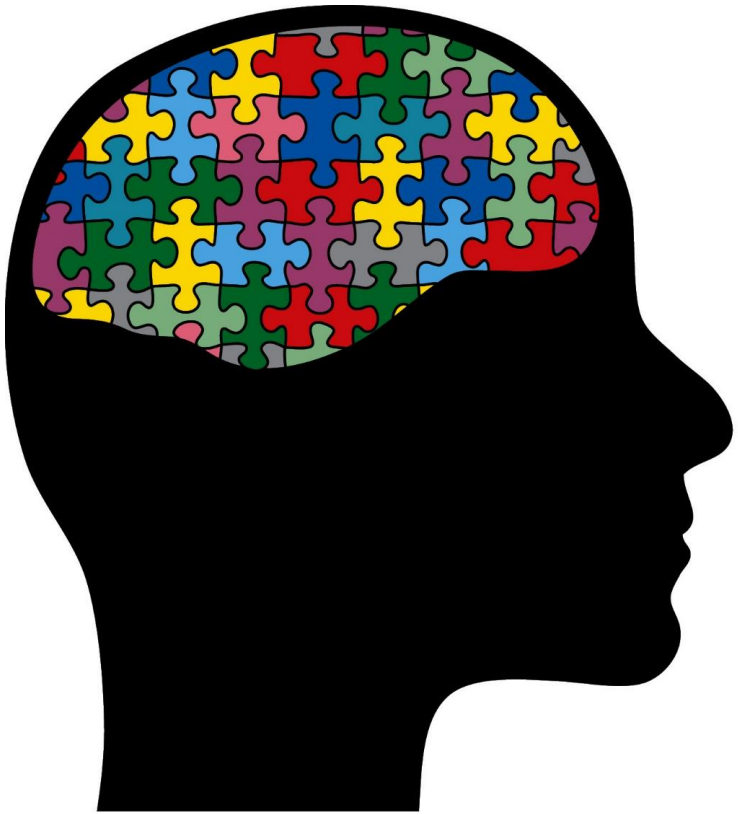
The function is implemented as follows:

```
def yolo_funtion(n):
        # sets are similar to lists but unordered
        # and faster to iterate through
        multiples = set()
        for i in range(2, n+1):
                if i not in multiples:
                print(i)
                # update is a set union
                multiples.update(range(i*i, n+1, i))

yolo_funtion(100)
```

Python has in-built datatypes and operations that make very compact algorithms, in less lines than C/C++ or Java.

INTRODUCTION TO
ARTIFICIAL NEURAL NETWORKS



What is an Artificial Neural Network (ANN)?

It is a mathematical model for structure, connectivity and signal propagation through animal neural tissue

What's so great about it?

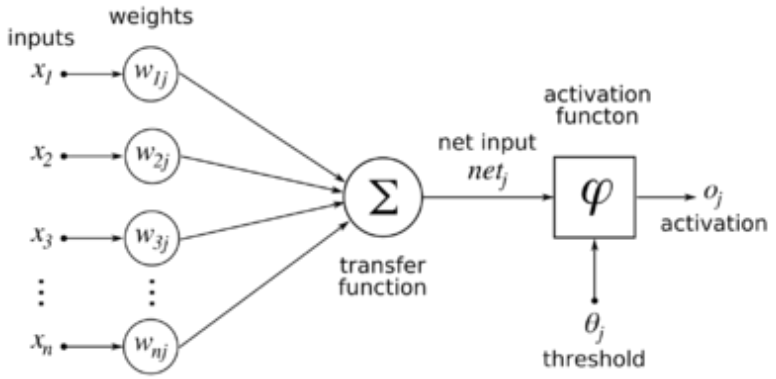
ANNs provide Scientists biological insight, however they can also be used as a computational model when combined with machine learning to solve problems in:

- Image and audio recognition
- Statistical Classification
- Time series forecasting
- Function approximation
- Control systems
- Robotics

ANNs typically have two parts, the network structure and the learning algorithm.

- A typical single artificial neuron sums a bunch of (usually normalized) real valued inputs applies a *transfer function* the inputs to get an output value.
- There are *weights* associated with each input
- The transfer function is typically nonlinear, like a sigmoid or *tanh(x)*

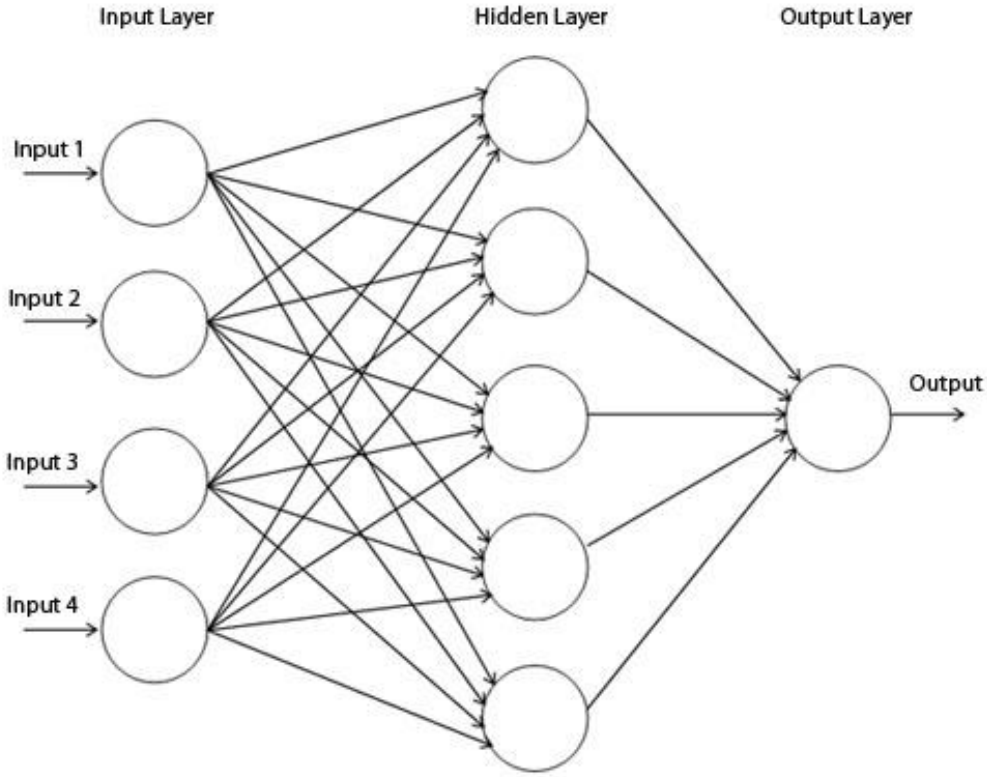
The learning algorithm updates the *weights* in the input layer of the NN in order to learn the correct data classification



Typical NNs are multilayered. There is one NN called the *Multi Layered Perceptron (MLP)* that is popularly used for classification, regression and pattern recognition and its structure consists of three types of layers:

- The *input layer* pushes input forward to the hidden layer
- The *hidden layer* can be multilayered
- The *output layer* receives data pushed forward by all the layers.

Note that there is a node to every node connectivity. There is no feedback of data into the network, it is strictly feedforward.



One of the most important points in the history of ANNs was the rediscovery of the Back-propagation (BP) algorithm in the late 70's for training MLPs.

- MLPs use a supervised learning technique called BP that minimizes the residual of the output versus expected training output
- We write the Neural Net as the following equation:

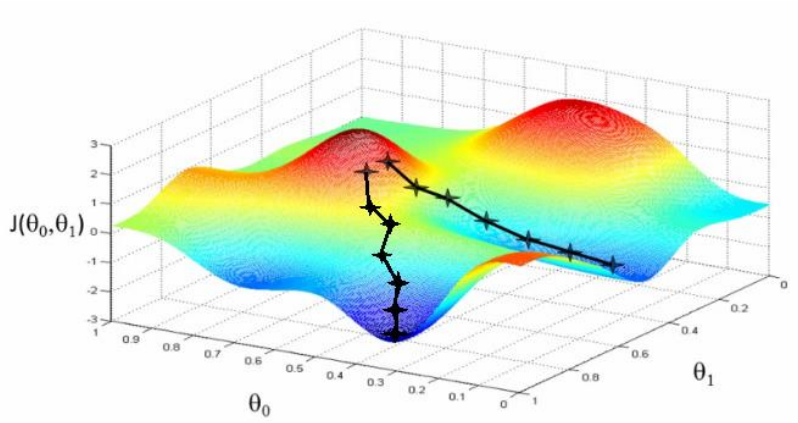
$$y = \sum_{i=0} \sum_{j=0} \phi_{ij}(w_{ij} x_{ij}) = output\ node$$

- This sums all the forward outputs x_{ij} in each layer and produces an output

We then minimize the Squared residual of the output y_i and the training data by taking its partial derivative with respect to each weight in a layer and

$$\frac{\partial}{\partial w_{ij}} \frac{1}{2} (y_t - y)^2 = 0$$

From this we can derive the weight update equation after each training sample:



$$\Delta w_{ij} = -\gamma \frac{\partial y}{\partial w_{ij}}$$

Where γ is a constant called the *learning rate*.

However, traditional BP suffers from a number of issues such as slow convergence and poor performance on noisy data and small training sets.

Despite this it is a great example of how a machine learning algorithm works.

That's all, folks. Stay tuned for more to come.