**Linear Transformation**

**Weights And Biases**

**Description**

Neural Networks have been making a lot of news lately. Different types of their architectures have led to incredible results, including beating 8-bit games, recognising images at scale, performing at par with humans at speech recognition, and beating the Japanese game of Go. A concept that was born in the '70s, neural nets have made an gigantic comeback in the last decade or so owing to tremendous improvement in the hardware to run these nets.  
  
In this question, we will try to demonstrate one of the two main computations that comprise of a neural network - matrix multiplication (the other computation is applying a nonlinear threshold at 0).  
  
Each layer of a neural network is made up of several "neurons", or in network terminology, nodes. Each node has an input and an output. Also, each node possesses a matrix of weights as well as a vector of biases.   
  
If 'w' is the matrix of weights, 'b' is the bias vector, and 'x' is the input into this node, the output is given by   
output = (w\*x + b)  
  
Following is the matrix of weights:  
w = [[0.42, -0.54, 0.56, -0.61, 0.58], [0.33, -0.53, 0.24, 0.75, -0.34], [0.34, -0.21, 0.11, 0.03, 0.78], [-0.34, -0.65, 0.93, -0.87, 0.12], [0.12, 0.23, 0.65, -0.47, -0.59]]  
  
Following is the vector of biases  
b = [2.10, 3.41, 4.54, 3.76, 0.29]  
  
Now, say the following table of inputs is given to the problem:  
x\_input = [3, 4, 6, 3, 7]  
  
Using the above formula, we are trying to find the output vector of this node. Run the code to see this in action!

import numpy as np

# Enter the values of the matrices

w = np.matrix([[0.42, -0.54, 0.56, -0.61, 0.58], [0.33, -0.53, 0.24, 0.75, -0.34], [0.34, -0.21, 0.11, 0.03, 0.78], [-0.34, -0.65, 0.93, -0.87, 0.12], [0.12, 0.23, 0.65, -0.47, -0.59]])

x = np.transpose(np.matrix([[2.10, 3.41, 4.54, 3.76, 0.29]]))

b = np.transpose(np.matrix([[3, 4, 6, 3, 7]]))

# Check the shapes to ensure the multiplication will go through

print ("Dimensions of weights: ", w.shape)

print ("Dimensions of inputs: ", x.shape)

print ("Dimensions of bias vector: ", b.shape)

# Formula for the output of a node

output = (w \* x) + b

print("The output vector of this layer is \n", output)

print ("Congratulations, you have performed a neural net computation by hand!")

**Linear Regression**

**Linear Regression - Matrix Multiplication**

**Description**

Linear Regression can almost be considered as the "Hello, world" of model-building. You will learn about this in a lot of detail in upcoming modules. For now, let's simply examine it purely as a linear algebra problem.  
  
Part of Linear Regression is what is called the estimation equation. This is expressed as:

X\*B = Y

# X - input variables matrix

# B - coefficient matrix

# Y - output variable matrix

As an example, let's consider [this small dataset](https://newonlinecourses.science.psu.edu/stat501/sites/onlinecourses.science.psu.edu.stat501/files/data/soapsuds/index.txt). It has two variables - one predictor (input), the other a response (output). Following are the input and output matrices:  
X = [[1,4], [1, 4.5], [1,5], [1, 5.5], [1, 6], [1, 6.5], [1, 7]]  
Y = [[33], [42], [45], [51], [53], [61], [62]]  
(The '1' values in the first column come from the fact that the input matrix has two coefficients, and the first one of those is not multiplied by x, and thus has coefficient 1)  
  
To calculate the coefficient matrix, the equation is given by

B = (inverse(X'X))\*(X'Y)

where X' is the transpose of matrix X

import numpy as np

# Create the matrix objects

X = np.matrix([[1.0, 4.0], [1.0, 4.5], [1.0, 5.0], [1.0, 5.5], [1.0, 6.0], [1.0, 6.5], [1.0, 7.0]])

print("The dimension of X are ", X.shape)

X\_prime = np.transpose(X)

print("The dimension of X transposed are ", X\_prime.shape)

# We need to calculate two matrices to find the coefficients matrix - inverse(X'X) and (X'Y)

# First, calculate inverse of (X'X)

# It is imperative that the (X'X) matrix is non-singular, otherwise its inverse will not exist.

X\_prime\_X = X\_prime \* X

X\_prime\_X\_inv = np.linalg.inv(X\_prime\_X)

# print (X\_prime\_X)

print ("The dimensions of X\_prime\_X\_inv are ", X\_prime\_X\_inv.shape)

# Now, moving on to the Y matrix. These are the output variables .

Y = np.transpose(np.matrix([[33, 42, 45, 51, 53, 61, 62]]))

# Calculate X'Y

X\_prime\_Y = X\_prime \* Y

# print(X\_prime\_Y)

print ("The dimensions of X\_prime\_Y are ", X\_prime\_Y.shape)

# Now, put them together to find B, the matrix of coefficients.

B = X\_prime\_X\_inv \* X\_prime\_Y

print ("The coefficient matrix is given by \n", B)

print ("Congratulations, you just calculated coefficients of a linear regression by hand!")

print ("Feel free to change around some numbers and run the code again!")

# This is exactly the mechanism used to calculate coefficients when a Python library "fits" a model.