

Get a data science job, guaranteed    [Click Here](#)

MAR 21, 2017

# A Beginner's Guide to Neural Networks in Python and SciKit Learn 0.18

👤 Jose Portilla    💬 0    📁 DATA SCIENCE ([HTTPS://WWW.SPRINGBOARD.COM/BLOG/CATEGORY/](https://www.springboard.com/blog/category/data-science/)  
[/DATA-SCIENCE/](https://www.springboard.com/blog/category/data-science/))



## Building a Neural Network in Python

I'm Jose Portilla and I **teach thousands of students on Udemy** (<https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/>) about Data Science and Programming and I also conduct **in-person programming and data science training** (<http://www.pieriandata.com/>), for more info you can reach me at **training AT pieriandata.com**.

The code and data for this tutorial is at [Springboard's blog tutorials repository](https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla_SpringBoard_Blog_Neural_Network.ipynb) ([https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla\\_SpringBoard\\_Blog\\_Neural\\_Network.ipynb](https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla_SpringBoard_Blog_Neural_Network.ipynb)), if you want to follow along.

The most popular machine learning library for Python is [SciKit Learn](http://scikit-learn.org/) (<http://scikit-learn.org/>). The latest version (0.18) now has built in support for Neural Network models! In this article we will learn how Neural Networks work and how to implement them with the Python programming language and the latest version of SciKit-Learn! Basic understanding of Python is necessary to understand this article, and it would also be helpful (but not necessary) to have some experience with Sci-Kit Learn.

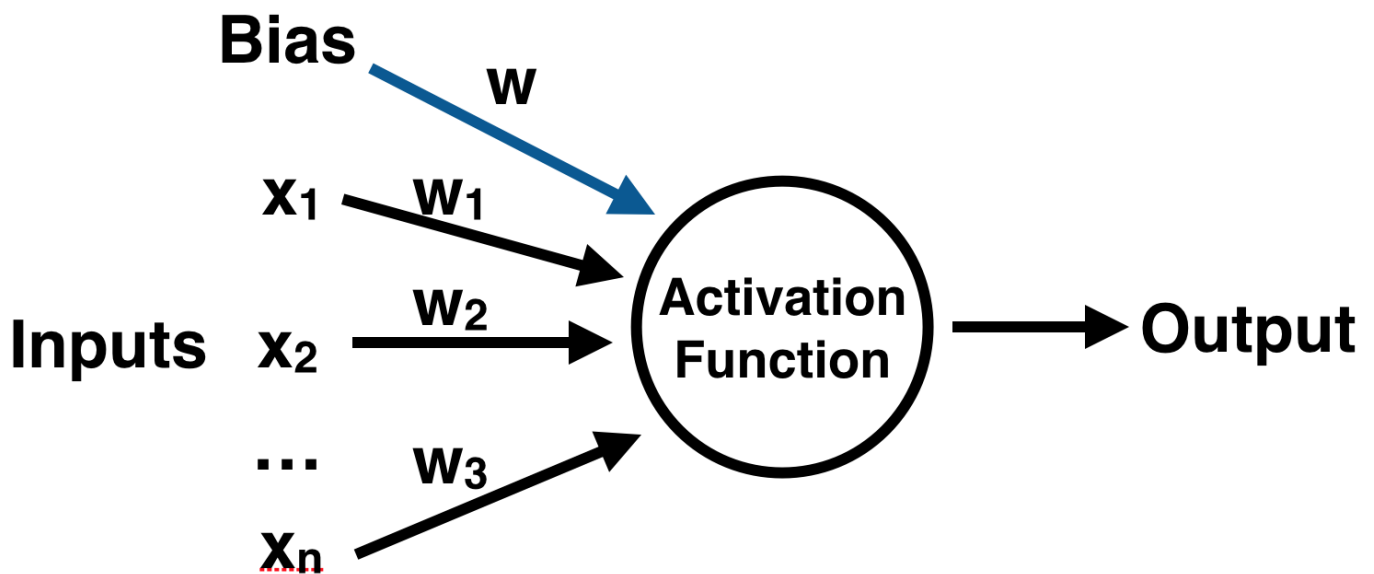
# Neural Networks

Neural Networks are a machine learning framework that attempts to mimic the learning pattern of natural biological neural networks: you can think of them as a crude approximation of what we assume the human mind is doing when it is learning. Biological neural networks have interconnected neurons with dendrites that receive inputs, then based on these inputs they produce an output signal through an axon to another neuron. We will try to mimic this process through the use of Artificial Neural Networks (ANN), which we will just refer to as neural networks from now on. Neural networks are the foundation of deep learning, a subset of machine learning that is responsible for some of the most exciting technological advances today! The process of creating a neural network in Python begins with the most basic form, a single perceptron. Let's start by explaining the single perceptron!

## The Perceptron

Let's start our discussion by talking about the Perceptron! A perceptron has one or more inputs, a bias, an activation function, and a single output. The perceptron receives inputs, multiplies them by some weight, and then passes them into an activation function to produce an output. There are many possible activation functions to choose from, such as the logistic function, a

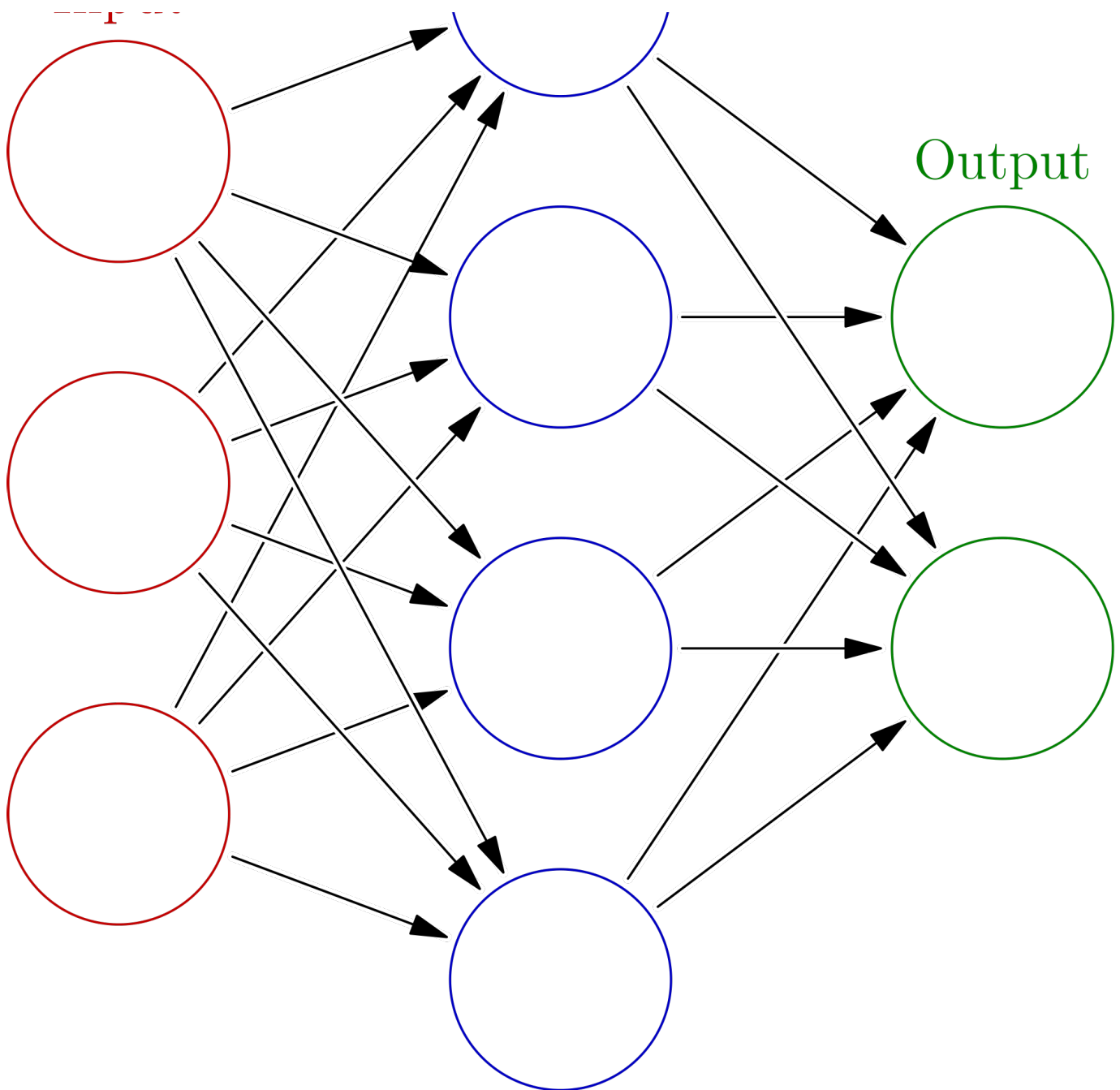
trigonometric function, a step function etc. We must also make sure to add a [bias to the perceptron](https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks) (<https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks>), a constant weight outside of the inputs that allows us to achieve better fit for our predictive models. Check out the diagram below for a visualization of a perceptron:



Once we have the output we can compare it to a known label and adjust the weights accordingly (the weights usually start off with random initialization values). We keep repeating this process until we have reached a maximum number of allowed iterations, or an acceptable error rate.

To create a neural network, we simply begin to add layers of perceptrons together, creating a multi-layer perceptron model of a neural network. You'll have an input layer which directly takes in your data and an output layer which will create the resulting outputs. Any layers in between are known as hidden layers because they don't directly "see" the feature inputs within the data you feed in or the outputs. For a visualization of this check out the diagram below (source: Wikipedia).





Keep in mind that due to their nature, neural networks tend to work better on GPUs than on CPU. The sci-kit learn framework isn't built for GPU optimization. If you want to continue using GPUs and distributed models, take a look at some other frameworks, such as Google's open sourced [TensorFlow](https://github.com/tensorflow/tensorflow) (<https://github.com/tensorflow/tensorflow>).

Let's move on to actually creating a neural network with Python and Sci-Kit Learn!

# SciKit-Learn

In order to follow along with this tutorial, you'll need to have the latest version of SciKit Learn (>0.18) installed! It is easily installable either through pip or conda, but you can reference the [official installation documentation](http://scikit-learn.org/stable/install.html) (<http://scikit-learn.org/stable/install.html>) for complete details on this.

# Anaconda and iPython Notebook

One easy way of getting SciKit-Learn and all of the tools you need to have to do this exercise is by using Anaconda's iPython Notebook software. [This tutorial](https://www.safaribooksonline.com/blog/2013/12/12/start-ipython-notebook/) (<https://www.safaribooksonline.com/blog/2013/12/12/start-ipython-notebook/>) will help you get started with these tools so you can build a neural network in Python within.

# Data

For this analysis we will cover one of life's most important topics – Wine! All joking aside, [wine fraud](https://en.wikipedia.org/wiki/Wine_fraud) ([https://en.wikipedia.org/wiki/Wine\\_fraud](https://en.wikipedia.org/wiki/Wine_fraud)) is a very real thing. Let's see if a Neural Network in Python can help with this problem! We will use the wine data set from the UCI Machine Learning Repository. It has various chemical features of different wines, all grown in the same region in Italy, but the data is labeled by three different possible cultivars. We will try to build a model that can classify what cultivar a wine belongs to based on its chemical features using Neural Networks. You can get the data [here](https://archive.ics.uci.edu/ml/datasets/Wine). (<https://archive.ics.uci.edu/ml/datasets/Wine>)

First let's import the dataset! We'll use the names feature of Pandas to make sure that the column names associated with the data come through.

In [11]:

```
import pandas as pd
wine = pd.read_csv('wine_data.csv', names = ["Cultivator", "Alchol", "Malic_Acid", "Ash",
"Alcalinity_of_Ash", "Magnesium", "Total_phenols", "Falvanoids", "Nonflavanoid_phenols",
"Proanthocyanins", "Color_intensity", "Hue", "OD280", "Proline"])
```

Let's check out the data:

In [9]:

```
wine.head()
```

Out[9] (we've cut some columns from this output in the interests of formatting for this blog post — you should see more):

	<b>Cultivator</b>	<b>Alchol</b>	<b>Malic_Acid</b>	<b>Ash</b>
<b>0</b>	1	14.23	1.71	2.43
<b>1</b>	1	13.20	1.78	2.14
<b>2</b>	1	13.16	2.36	2.67
<b>3</b>	1	14.37	1.95	2.50
<b>4</b>	1	13.24	2.59	2.87

In [12]:

```
wine.describe().transpose()
```

Out[12] (we've cut the standard deviation (std) and count columns from this output in the interests of formatting for this blog post):

	<b>mean</b>	<b>min</b>	<b>25%</b>	<b>50%</b>	<b>75%</b>	<b>max</b>
<b>Cultivator</b>	1.938202	1.00	1.0000	2.000	3.0000	3.00
<b>Alchol</b>	13.000618	11.03	12.3625	13.050	13.6775	14.83
<b>Malic_Acid</b>	2.336348	0.74	1.6025	1.865	3.0825	5.80
<b>Ash</b>	2.366517	1.36	2.2100	2.360	2.5575	3.23

	mean	min	25%	50%	75%	max
<b>Alcalinity_of_Ash</b>	19.494944	10.60	17.2000	19.500	21.5000	30.00
<b>Magnesium</b>	99.741573	70.00	88.0000	98.000	107.0000	162.00
<b>Total_phenols</b>	2.295112	0.98	1.7425	2.355	2.8000	3.88
<b>Falvanoids</b>	2.029270	0.34	1.2050	2.135	2.8750	5.08
<b>Nonflavanoid_phenols</b>	0.361854	0.13	0.2700	0.340	0.4375	0.66
<b>Proanthocyanins</b>	1.590899	0.41	1.2500	1.555	1.9500	3.58
<b>Color_intensity</b>	5.058090	1.28	3.2200	4.690	6.2000	13.00
<b>Hue</b>	0.957449	0.48	0.7825	0.965	1.1200	1.71
<b>OD280</b>	2.611685	1.27	1.9375	2.780	3.1700	4.00
<b>Proline</b>	746.893258	278.00	500.5000	673.500	985.0000	1680.00

In [13]:

```
# 178 data points with 13 features and 1 label column  
wine.shape
```

Out[13]:

```
(178, 14)
```

Let's set up our Data and our Labels:

In [14]:

```
X = wine.drop('Cultivator',axis=1)  
y = wine['Cultivator']
```

## Train Test Split

Let's split our data into training and testing sets, this is done easily with SciKit Learn's `train_test_split` function from `model_selection`:

In [15]:

```
from sklearn.model_selection import train_test_split
```

In [16]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

## Data Preprocessing

The neural network in Python may have difficulty converging before the maximum number of iterations allowed if the data is not normalized. Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data. Note that you must apply the same scaling to the test set for meaningful results. There are a lot of different methods for normalization of data, we will use the built-in `StandardScaler` for standardization.

In [17]:

```
from sklearn.preprocessing import StandardScaler
```

In [18]:

```
scaler = StandardScaler()
```

In [19]:

```
# Fit only to the training data  
scaler.fit(X_train)
```



Out[19]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [20]:

```
# Now apply the transformations to the data:  
X_train = scaler.transform(X_train)  
X_test = scaler.transform(X_test)
```

## Training the model

Now it is time to train our model. SciKit Learn makes this incredibly easy, by using estimator objects. In this case we will import our estimator (the Multi-Layer Perceptron Classifier model) from the `neural_network` library of SciKit-Learn!

In [21]:

```
from sklearn.neural_network import MLPClassifier
```

Next we create an instance of the model, there are a lot of parameters you can choose to define and customize here, we will only define the `hidden_layer_sizes`. For this parameter you pass in a tuple consisting of the number of neurons you want at each layer, where the `nth` entry in the tuple represents the number of neurons in the `nth` layer of the MLP model. There are many ways to choose these numbers, but for simplicity we will choose 3 layers with the same number of neurons as there are features in our data set along with 500 max iterations.

In [24]:

```
mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)
```

Now that the model has been made we can fit the training data to our model, remember that this data has already been processed and scaled:

In [25]:

```
mlp.fit(X_train,y_train)
```

Out[25]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(13, 13, 13), learning_rate='constant',
              learning_rate_init=0.001, max_iter=500, momentum=0.9,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
```

You can see the output that shows the default values of the other parameters in the model. I encourage you to play around with them and discover what effects they have on your neural network in Python!

## Predictions and Evaluation

Now that we have a model it is time to use it to get predictions! We can do this simply with the `predict()` method off of our fitted model:

In [26]:

```
predictions = mlp.predict(X_test)
```

Now we can use SciKit-Learn's built in metrics such as a classification report and confusion matrix to evaluate how well our model performed:

In [27]:

```
from sklearn.metrics import classification_report, confusion_matrix
```

In [28]:

```
print(confusion_matrix(y_test, predictions))
```

```
[[17  0  0]
 [ 0 14  1]
 [ 0  0 13]]
```

In [29]:

```
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
1	1.00	1.00	1.00	17
2	1.00	0.93	0.97	15
3	0.93	1.00	0.96	13
avg / total	0.98	0.98	0.98	45

Not bad! Looks like we only misclassified one bottle of wine in our test data! This is pretty good considering how few lines of code we had to write for our neural network in Python. The downside however to using a Multi-Layer Perceptron model is how difficult it is to interpret the model itself. The weights and biases won't be easily interpretable in relation to which features are important to the model itself.

However, if you do want to extract the MLP weights and biases after training your model, you use its public attributes **coefs\_** and **intercepts\_**.

**coefs\_** is a list of weight matrices, where weight matrix at index  $i$  represents the weights between layer  $i$  and layer  $i+1$ .

**intercepts\_** is a list of bias vectors, where the vector at index  $i$  represents the bias values added to layer  $i+1$ .

In [30]:

```
len(mlp.coefs_)
```

Out[30]:

```
4
```

In [31]:

```
len(mlp.coefs_[0])
```

Out[31]:

```
13
```

In [32]:

```
len(mlp.intercepts_[0])
```

Out[32]:

```
13
```

# Conclusion

Hopefully you've enjoyed this brief discussion on Neural Networks! Try playing around with the number of hidden layers and neurons and see how they effect the results of your neural network in Python!

Want to learn more? You can check out my Python for Data Science and Machine Learning course on Udemy! Get it for 90% off at this link: <https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/> (<https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/>)

If you are looking for corporate in-person training, feel free to contact me at: training AT pieriandata.com.

Please feel free to follow along with the code [here](https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla_SpringBoard_Blog_Neural_Network.ipynb) ([https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla\\_SpringBoard\\_Blog\\_Neural\\_Network.ipynb](https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Neural%20Networks%20/JMPortilla_SpringBoard_Blog_Neural_Network.ipynb)) and leave comments below if you have any questions!

LEARN UX DESIGN WITH SPRINGBOARD

LEARN DATA SCIENCE WITH SPRINGBOARD



(<https://www.springboard.com/workshops/ux->

design?utm\_source=blog&utm\_medium=uxsidebar&  
utm\_campaign=uxblogsidebar)

Related Articles

What is data science? A data science  
definition.  
(https://www.springboard.com  
/blog/data-science-definition/)

LEARN DATA ANALYSIS WITH SPRINGBOARD



Data Science Career Track  
Five Awesome Data Science Projects  
Master data science skills, and get  
a job or your money back  
(https://www.springboard.com

/blog/data-science-projects-bootcamp-  
(https://www.springboard.com/workshops/data-  
science-career-track?utm\_source=blog&  
utm\_medium=dssidebar&  
utm\_campaign=dsblogsidebar)

AUTHOR

Jose Portilla  
(https://www.springboard.com  
/blog/author/josemarcial/)



(https://www.springboard.com

/blog/author/josemarcial/)



(https://www.springboard.com/workshops

/analytics?utm\_source=blog&  
utm\_medium=dasidebar&

9 Comments [Springboard Blog](#)

1 Login ▾

♥ Recommend 1 [Share](#)

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)



Name



**Steve Maughan** • 7 months ago

It would have been helpful to provide the data file you used. The link you provide doesn't have the data in a nice csv format. It's a header-less csv text file.

2 ^ | ▾ • Reply • Share >



**Harminder Lall** • 2 months ago

Very helpful tutorial. I renamed the files to .csv and it worked.

^ | ▾ • Reply • Share >



**bani talwadia** • 3 months ago

The tutorial was very helpful.

I have a few questions though:

- 1) How to decide the optimum value for hidden layer parameters and alpha?
- 2) Before using ANN, do we need to do some prep work using statistical methods like removing statistically insignificant variables? Or the method you described above is sufficient and we need not do any prep work?
- 3) Can this technique be applied to a relatively larger data set which suppose has 60 independent variables or is there any other more efficient technique?

^ | ▾ • Reply • Share >



**Naveen Kashyap** • 4 months ago

Some body please help me. I am .net developer but facing problem to learn neural network. All go above my head. But i am very keen to learn this. I've installed python in Visual Studio and is working fine. Can somebody help me from where should i start to learn neural network as i have no significant knowledge about python and NN but i am very curious about training the models etc.

Any help will be appreciated.

Thanks

^ | ▾ • Reply • Share >

Millions of students are improving their skills by using our courses

LEARN MORE (/workshops/)

# Stay in the know!

I Want To Learn... ▼

Enter your email

GET UPDATES

- Workshops ▼
- Resources ▼
- Company ▼

(https://www.facebook.com/springboard)