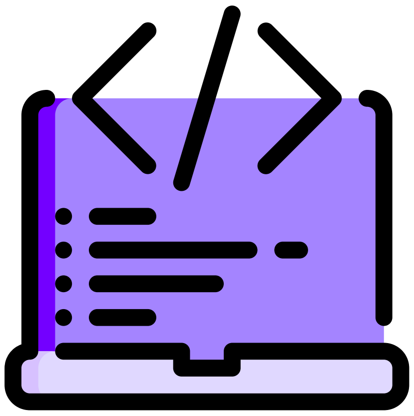
Building a

NEURAL NET FROM SCRATCH



What is a neural network?

Neural networks are one of the main tools used in machine learning. As **neural** suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn. NNs consist of input and output layers, as well as a hidden layer consisting of units that transform the input.

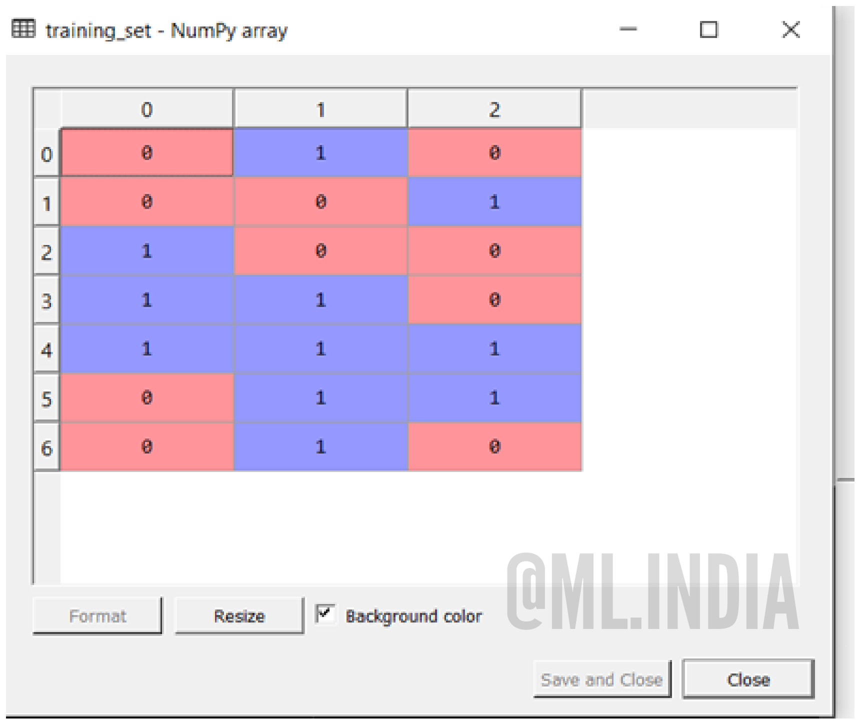
They are excellent tools for **finding patterns** which are far too complex or numerous for a human programmer to extract and teach the machine to recognize.

Steps for this mini-tutorial:

- Define independent and dependent variables.
- Define hyper-parameters.
- Define activation function and the derivative.
- Train the model.
- Make predictions.

Step #1: Variables.

```
#training data independent variable (x)
training_set = np.array([[0,1,0], #3 features , 7 entries
                         [0,0,1],
                         [1,0,0],
                         [1,1,0],
                         [1,1,1],
                         [0,1,1],
                         [0,1,0]])

#training data dependent variable (y)
labels = np.array([[1,
                    0,
                    0,
                    1,
                    1,
                    0,
                    1]])

#reshaping our dependent variable
labels = labels.reshape(7,1)
```



	0	1	2
0	0	1	0
1	0	0	1
2	1	0	0
3	1	1	0
4	1	1	1
5	0	1	1
6	0	1	0

Our **input set** contains seven records. Similarly, we also created a **labels set** that contains corresponding labels for each record in the input set. The labels are the values that we want our ANN to predict.

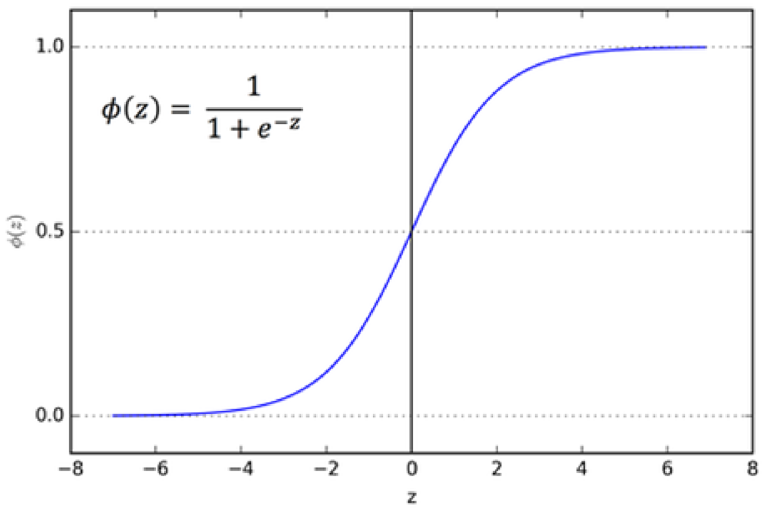
Step #2: Hyper-parameters.

Here **Random Seed** helps get the same values upon recursive execution and **Lr** is the learning rate; lr is the **step size** at each iteration while moving towards a minimum of a loss function.


```
#hyperparameters
np.random.seed(42)
weights = np.random.rand(3,1)
bias = np.random.rand(1)
lr = 0.05
```

Step #3: Activation function.

The **sigmoid function** returns 0.5 when the input is 0. It returns a value close to 1 if the input is a large positive number. In the case of negative input, the sigmoid function outputs a value close to zero.



Quick tip: Scroll down on our page to find a dedicated post on '**Sigmoid Function**', the pros and cons!

```
#methods
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x)*(1-sigmoid(x))
```

Step #4: Training!

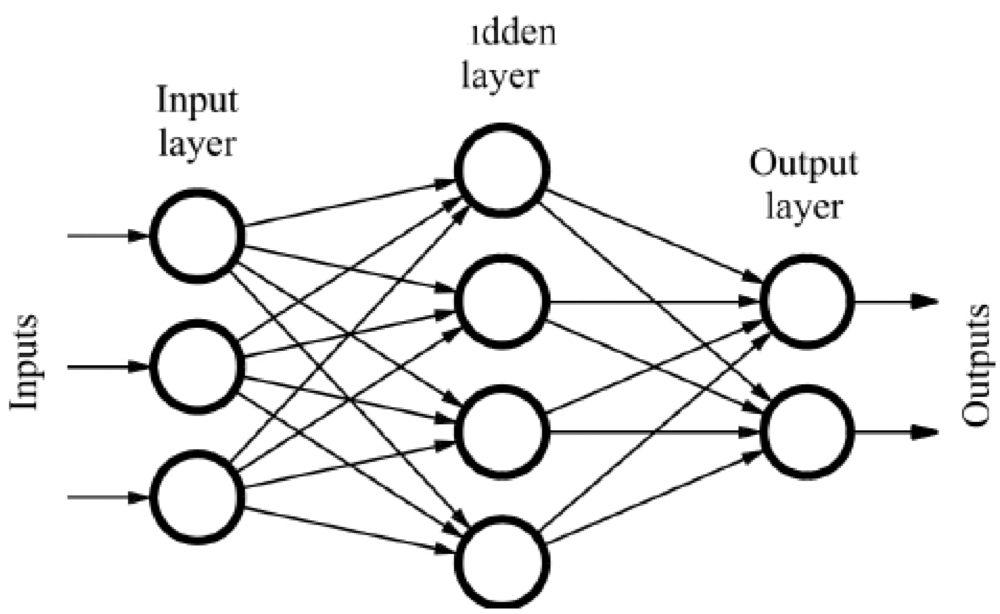
In the context of machine learning, an epoch is one complete pass through the training data. A deep neural network has to be trained for multiple epochs.


```
#trainging our model
for epoch in range(30000):
        inputs = training_set
        XW = np.dot(inputs, weights)+ bias
        z = sigmoid(XW)
        error = z - labels
        print(error.sum())
        dcost = error
        dpred = sigmoid_derivative(z)
        z_del = dcost * dpred
        inputs = training_set.T
        weights = weights - lr*np.dot(inputs, z_del)
        for num in z_del:
                bias = bias - lr*num

inputs = training_set
```

```
-0.0035377735791522064
-0.0035379953849952878
-0.0035382170289406795
-0.003538438511069622
-0.003538659831465743
-0.0035388809902110096
-0.0035391019873885765
-0.003539322282308074
-0.003539543497369263
-0.0035397640103365796
-0.003539984362065096
-0.003540204552637316
-0.003540424458213516
-0.0035406444506408397
-0.0035408641582373584
-0.003541083705005346
-0.003541303091028722
-0.0035415223163880183
-0.0035417413811650172
-0.0035419602854423887
-0.003542179029301734
-0.003542397612825321
```

Coding a **feed-forward neural network**:

```
#feed forward
XW = np.dot(inputs, weights)+ bias
z = sigmoid(XW)

#error
error = z - labels
print(error.sum())

#determining slope
slope = inputs * dcost * dpred

dcost = error
dpred = sigmoid_derivative(z)
z_del = dcost * dpred
inputs = training_set.T
weights = weights-lr*np.dot(inputs, z_del)

for num in z_del:
        bias = bias - lr*num
```



Step #5: Outcomes.

In the first case the output (result) is closer to 0, so will be classified as 0. Second one has the value closer to 1 , so will be classified as 1.

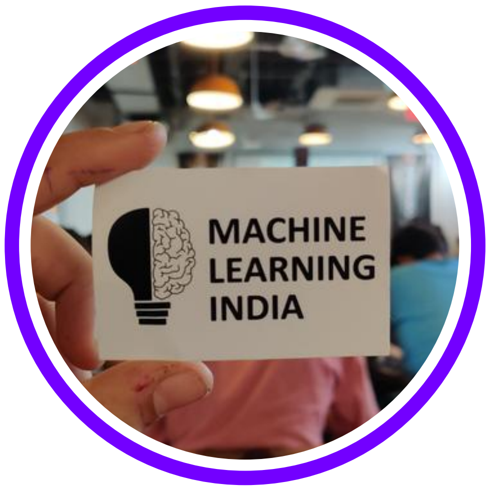

```
#predicting outcomes
single_pt = np.array([1,0,0])
result = sigmoid(np.dot(single_pt, weights) + bias)
print(result) #[0.051635]

single_pt = np.array([0,1,0])
result = sigmoid(np.dot(single_pt, weights) + bias)
print(result) #[0.99868149]
```

Other references:

- Build an Artificial Neural Network From Scratch: Part 1 by **Nagesh Singh Chauhan**, on **KDNuggets**.
- What is an artificial neural network? Here's everything you need to know by **Luke Dormehl** on **DigitalTrends**.

Important note:

The links to these resources will be put up on our Telegram. Channel ID: **@machinelearning24x7**.



Learnt something new?

Let us know in the comments! If you like our content and find it **valuable**, do give us a **follow**! Your **love** and **support** inspires us to keep delivering the best we can! ❤