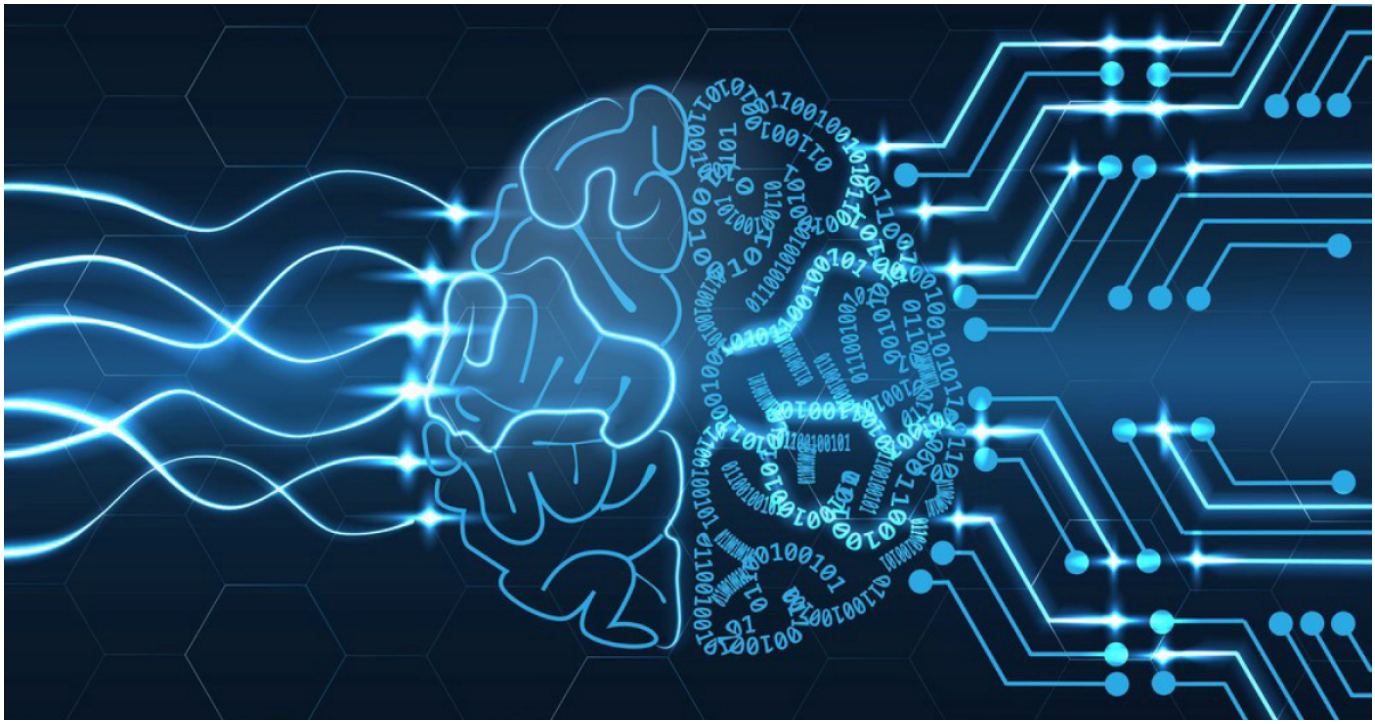
Building Blocks of Neural Network

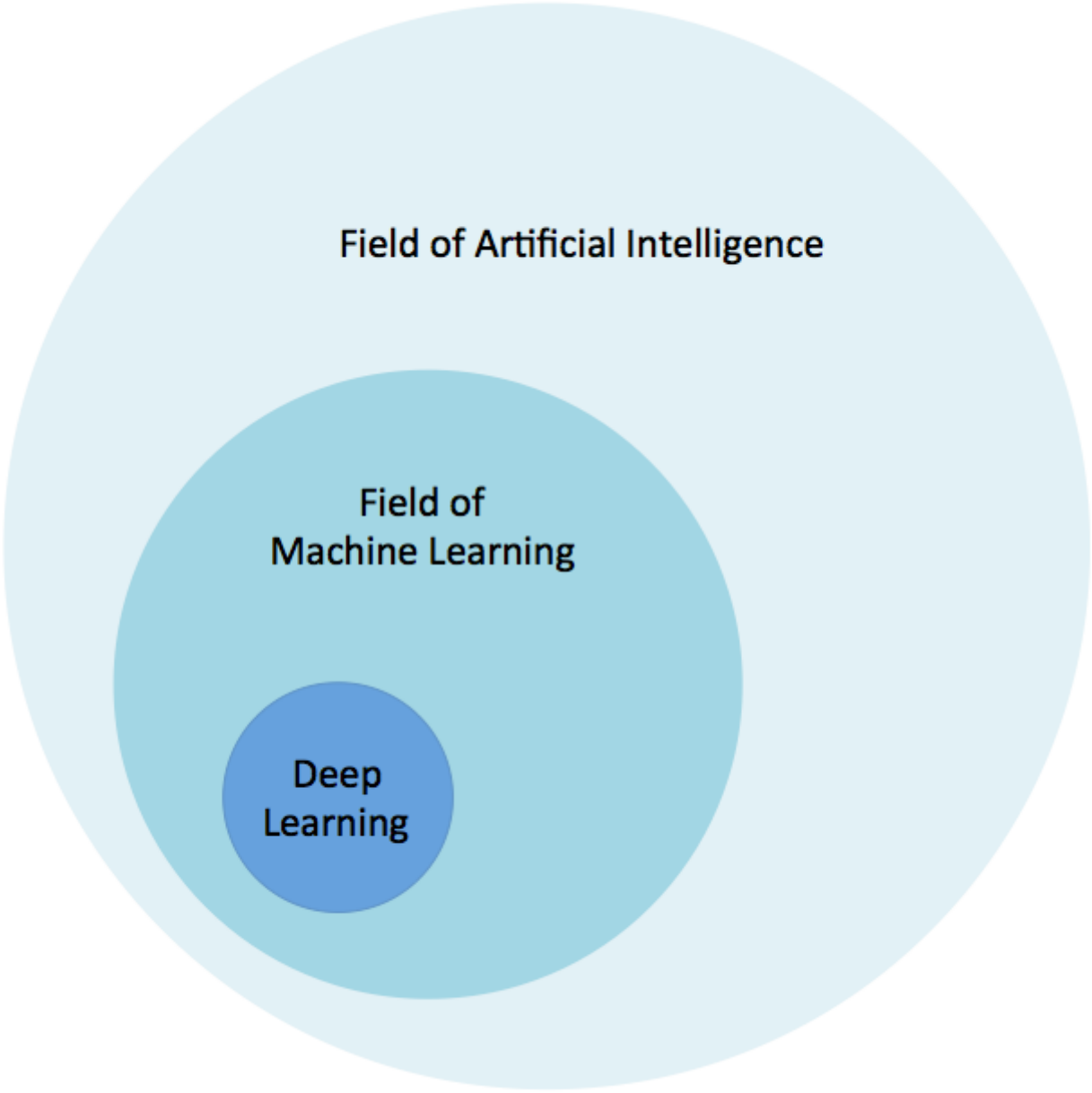


What is Deep Learning?

Watch the video inorder to understand what exactly deep learning is.

To understand what deep learning is, we first need to understand the relationship deep learning has with machine learning, neural networks and artificial intelligence

The best way to think of this relationship is to visualize them as cocentric circles:



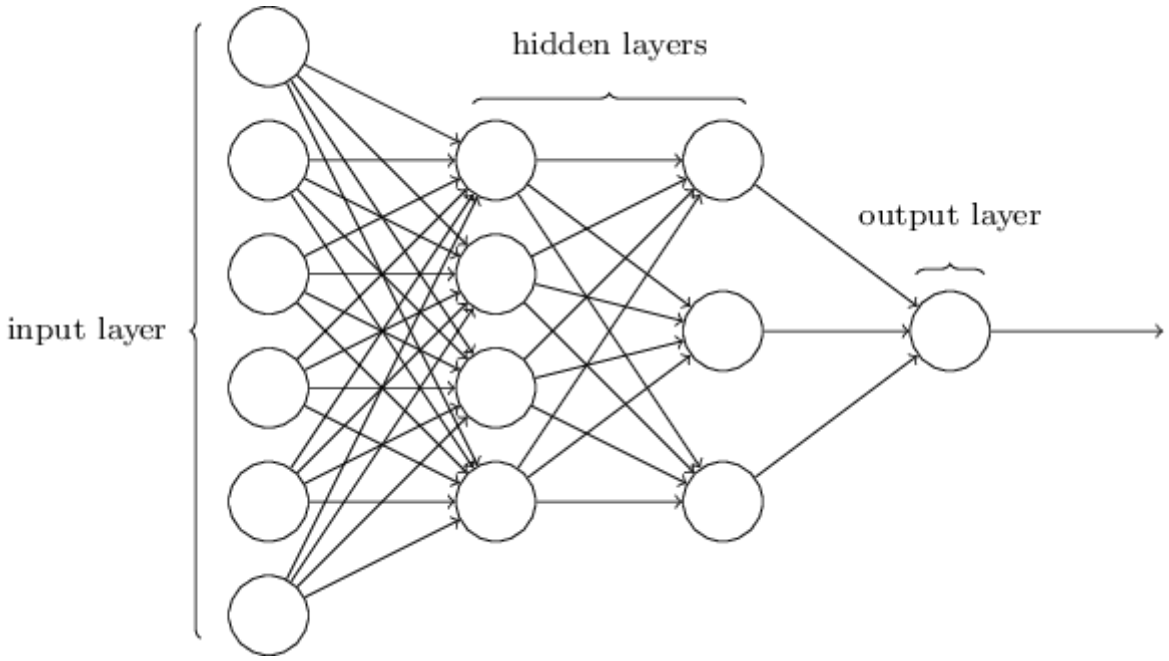
At the outer most ring you have artificial intelligence (using computers to reason). On layer inside of that is machine learning. With artificial neural networks and deep learning at the center.

Broadly speaking, deep learning is a more approachable name for an artificial neural network. The "deep" in deep learning refers to the depth of the network. An artificial neural network can be very shallow.

Neural networks are inspired by the structure of the cerebral cortex. At the basic level is the perceptron, the mathematical representation of a biological neuron. Like in the cerebral cortex, there can be several layers of interconnected perceptrons.

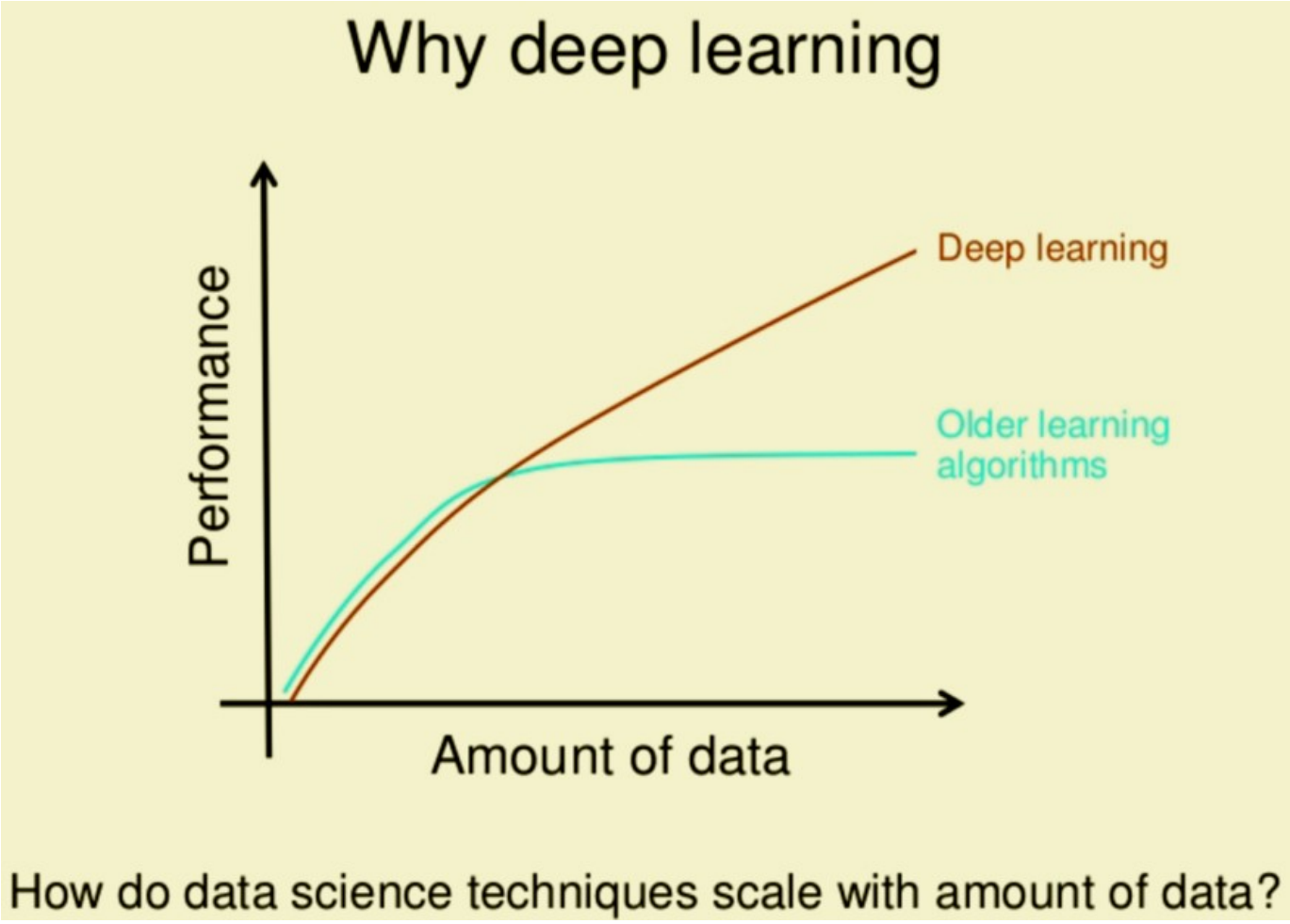
The first layer is the input layer. Each node in this takes an input, and then passes its output as the input to each node in the next layer. There are generally no connections between nodes in the same layer and the last layer and the last layer produces the outputs.

We call the middle part i.e., the hidden layer. These neurons have no connection to the outside (e.g. input or output) and are only activated by nodes in the previous layer.



Think of the deep learning as the technique for learning in neural networks that utilizes multiple layers of abstraction to solve pattern recognition problems. In 1980s, most neural netwworks were a single layer due to the cost of computation and availabilty of data.

Why is Deep Learning Important?



Computers have long had techniques for recognizing features inside of images. The results weren't always great. Computer vision has been a main beneficiary of deep learning. Computer vision using deep learning now rivals humans on many image recognition tasks.

Facebook has had great success with identifying faces in photographs by using deep learning. It's not just a marginal improvement, but a game changer: "Asked whether two unfamiliar photos of faces show the same person, a human being will get it right 97.53 percent of the time. New software developed by researchers at Facebook can score 97.25 percent on the same challenge, regardless of variations in lighting or whether the person in the picture is directly facing the camera."

Speech recognition is a another area that's felt deep learning's impact. Spoken languages are so vast and ambiguous. Baidu – one of the leading search engines of China – has developed a voice recognition system that is faster and more accurate than humans at producing text on a mobile phone. In both English and Mandarin.

What is particularly fascinating, is that generalizing the two languages didn't require much additional design effort: "Historically, people viewed Chinese and English as two vastly different languages, and so there was a need to design very different features," Andrew Ng says, chief scientist at Baidu. "The learning algorithms are now so general that you can just learn."

Google is now using deep learning to manage the energy at the company's data centers. They've cut their energy needs for cooling by 40%. That translates to about a 15% improvement in power usage efficiency for the company and hundreds of millions of dollars in savings.

Open Source Deep Learning Frameworks

Deep learnings is made accessible by a number of open source projects. Some of the most popular technologies include, but are not limited to, Deeplearning4j (DL4j), Theano, Torch, TensorFlow, and Caffe. The deciding factors on which one to use are the tech stack they target, and if they are low-level, academic, or application focused. Here's an overview of each:

DL4J:

```
JVM-based
Distrubted
Integrates with Hadoop and Spark
```

Theano:

```
Very popular in Academia
Fairly low level
Interfaced with via Python and Numpy
```

Torch:

```
Lua based
In house versions used by Facebook and Twitter
Contains pretrained models
```

TensorFlow:

```
Google written successor to Theano
Interfaced with via Python and Numpy
Highly parallel
Can be somewhat slow for certain problem sets
```

Caffe:

```
Not general purpose. Focuses on machine-vision problems
Implemented in C++ and is very fast
Not easily extensible
Has a Python interface
```

When to use Deep Learning?

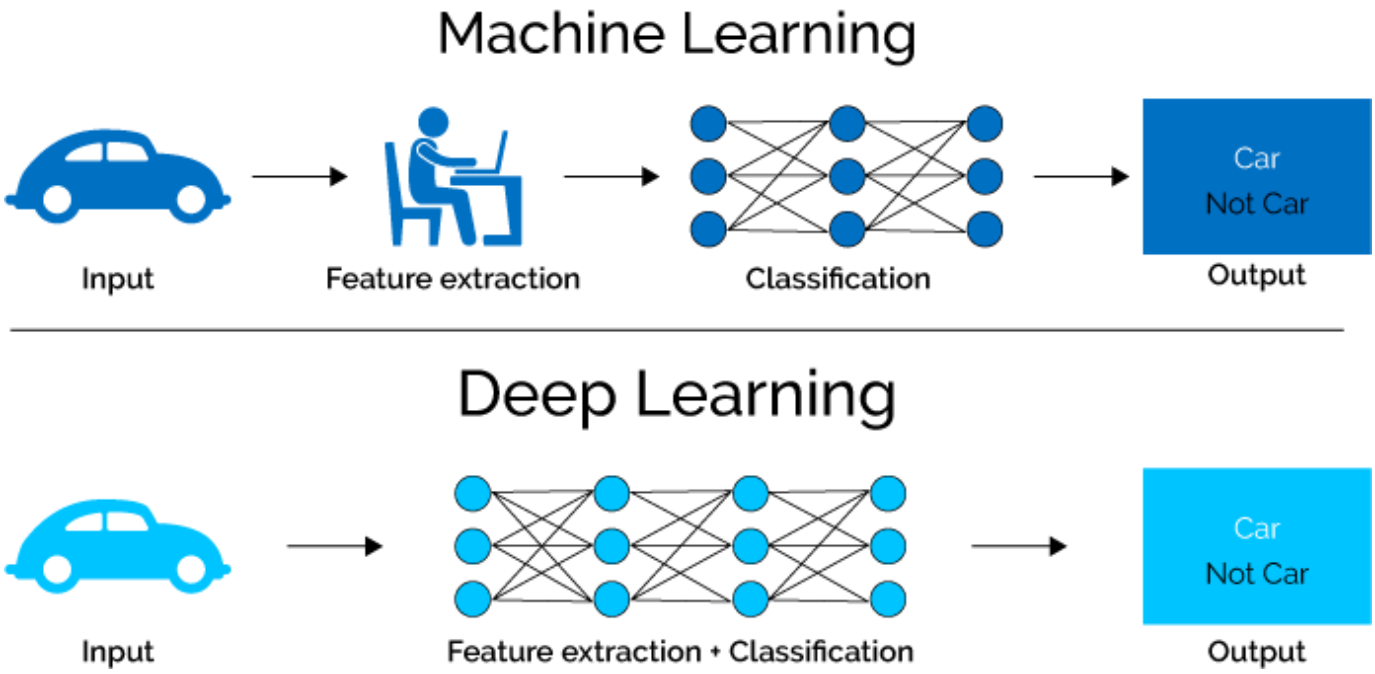
1. Deep Learning out perform other techniques if the data size is large. But with small data size, traditional Machine Learning algorithms are preferable.
2. Deep Learning techniques need to have high end infrastructure to train in reasonable time.
3. When there is lack of domain understanding for feature introspection, Deep Learning techniques outshines others as you have to worry less about feature engineering.
4. Deep Learning really shines when it comes to complex problems such as image classification, natural language processing, and speech recognition

Neural Networks

We have heard a lot about deep learning but before going through it, you must be wondering we already have so many machine learning algorithms then;

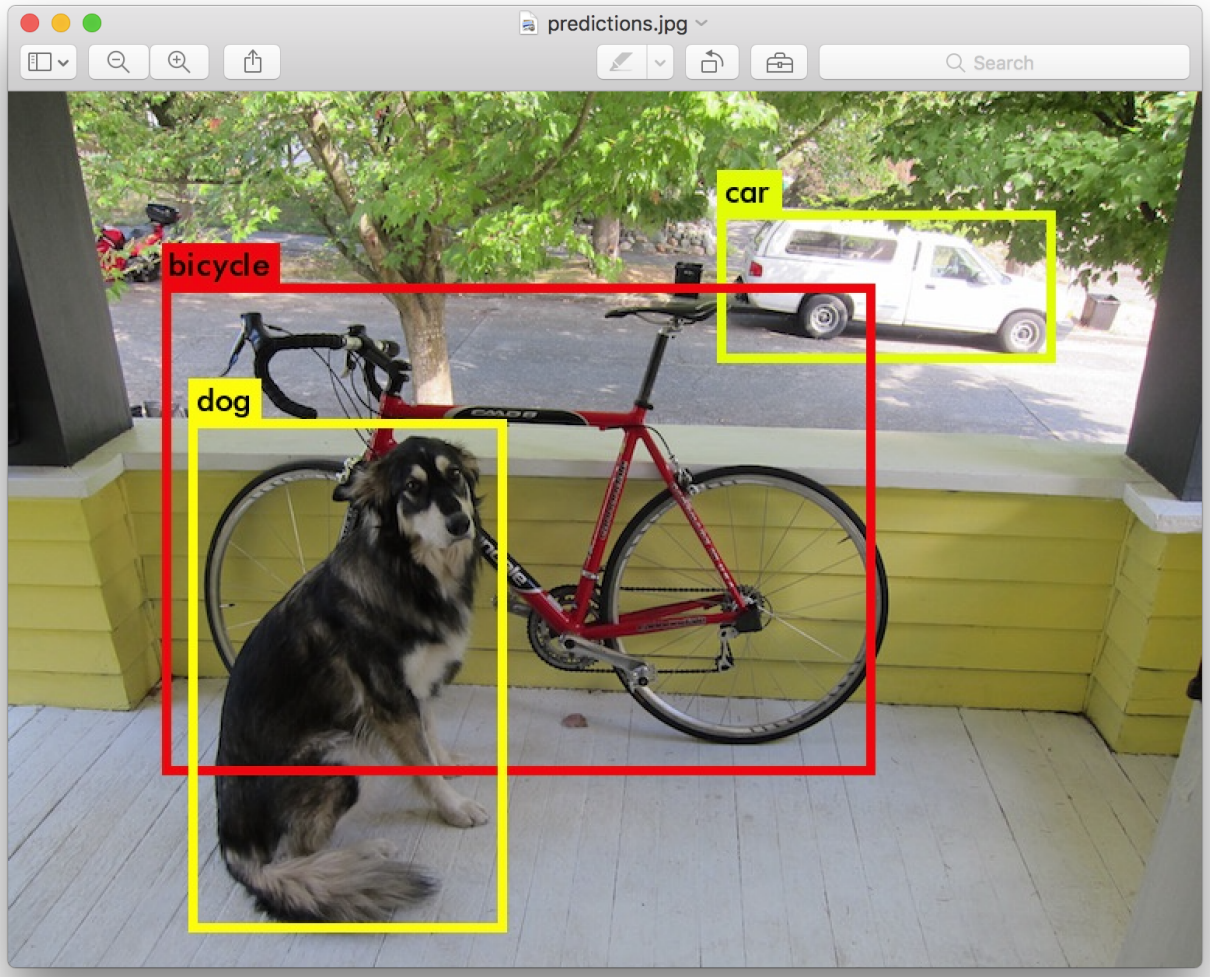
Why are neural nets preferred over traditional machine learning algorithms ?

In traditional Machine learning techniques, most of the applied features need to be identified by an domain expert in order to reduce the complexity of the data and make patterns more visible to learning algorithms to work. The biggest advantage Deep Learning algorithms as discussed before are that they try to learn high-level features from data in an incremental manner. This eliminates the need of domain expertise and hard core feature extraction.



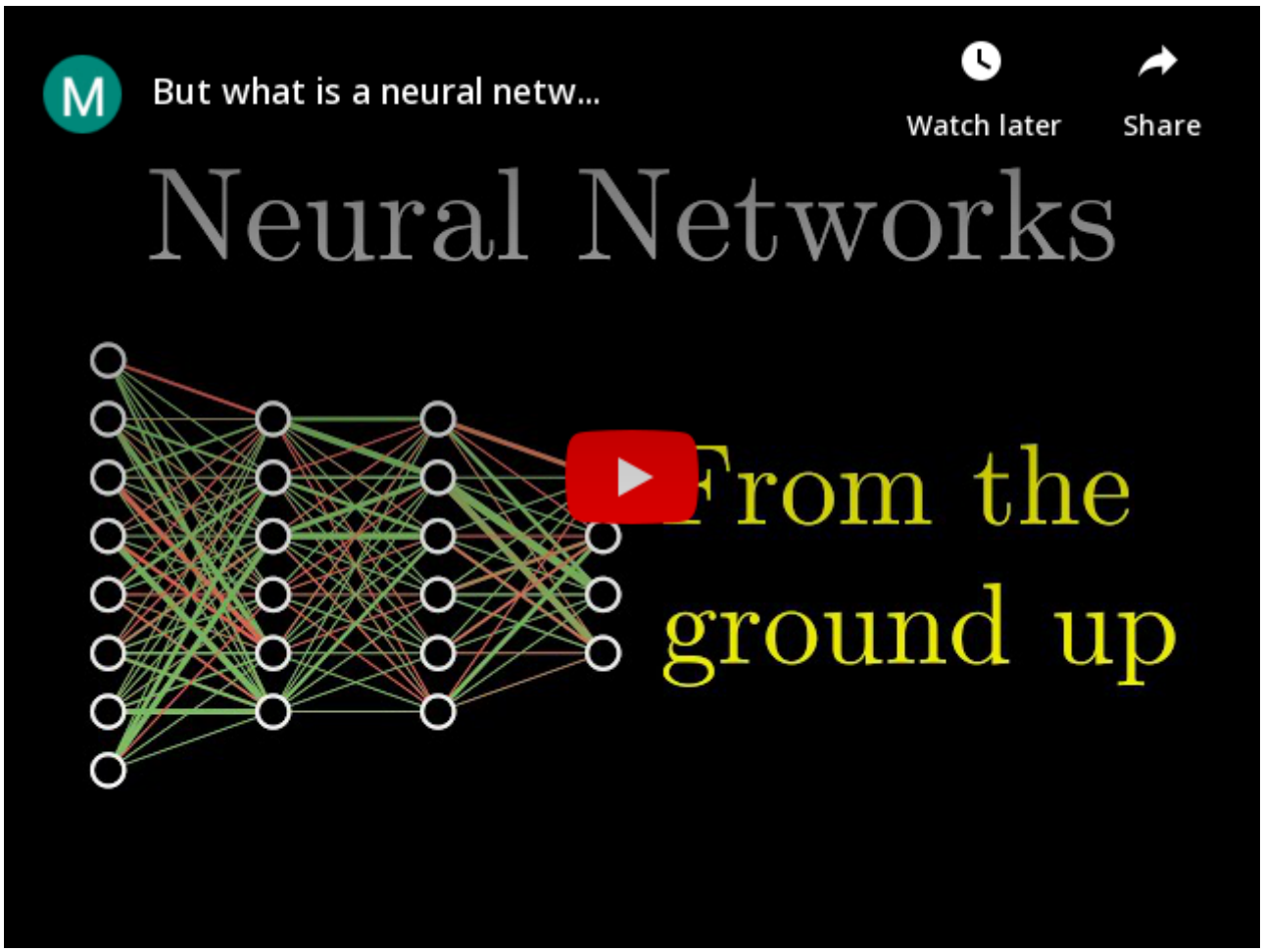
Another major difference between Deep Learning and Machine Learning technique is the problem solving approach. Deep Learning techniques tend to solve the problem end to end, where as Machine learning techniques need the problem statements to break down to different parts to be solved first and then their results to be combine at final stage. For example for a

multiple object detection problem, Deep Learning techniques like Yolo net take the image as input and provide the location and name of objects at output. But in usual Machine Learning algorithms like SVM, a bounding box object detection algorithm is required first to identify all possible objects to have the HOG as input to the learning algorithm in order to recognize relevant objects.



What is Neural Network?

Watch the video inorder to understand what neural networks actually are.



Neural networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning.

Human Brain is the most wonderful entity and it's really amazing to understand how it works. Our brain has the ability to learn everything itself on the basis of its own learning algorithm which led to an idea of imitating the human brain and creating Neural networks whose working principles are based on it.

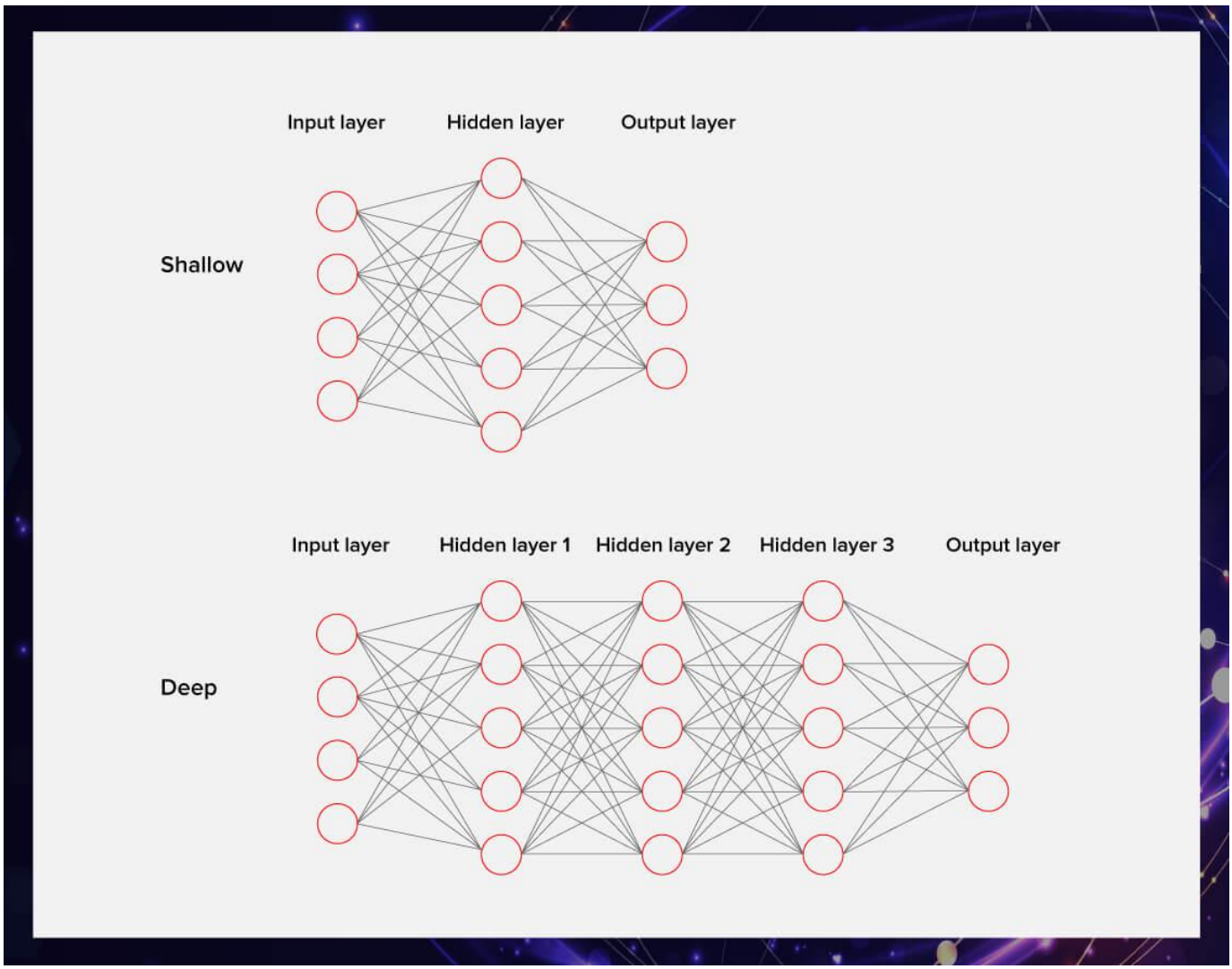
Recently in the field of computer science, neural networks has attracted a great deal of attention from many people. From doing various tasks like face recognition and speech recognition, to Healthcare and Marketing, Neural networks are widely used today.

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms.An artificial Neural network is the functional unit of Deep learning which is itself a part of Machine learning which itself is a subfield of Artificial Intelligence.

Neural networks were developed as simulating neurons or networks of neurons in the brain. Neurons have input wires called Dendrites which receive information from various locations. A neuron also has an output wire called Axon which sends messages to other neurons. Similarly,

an artificial Neural networks function when an input data is provided to it. Then this data is processed via layers of perceptrons to give a desired output.

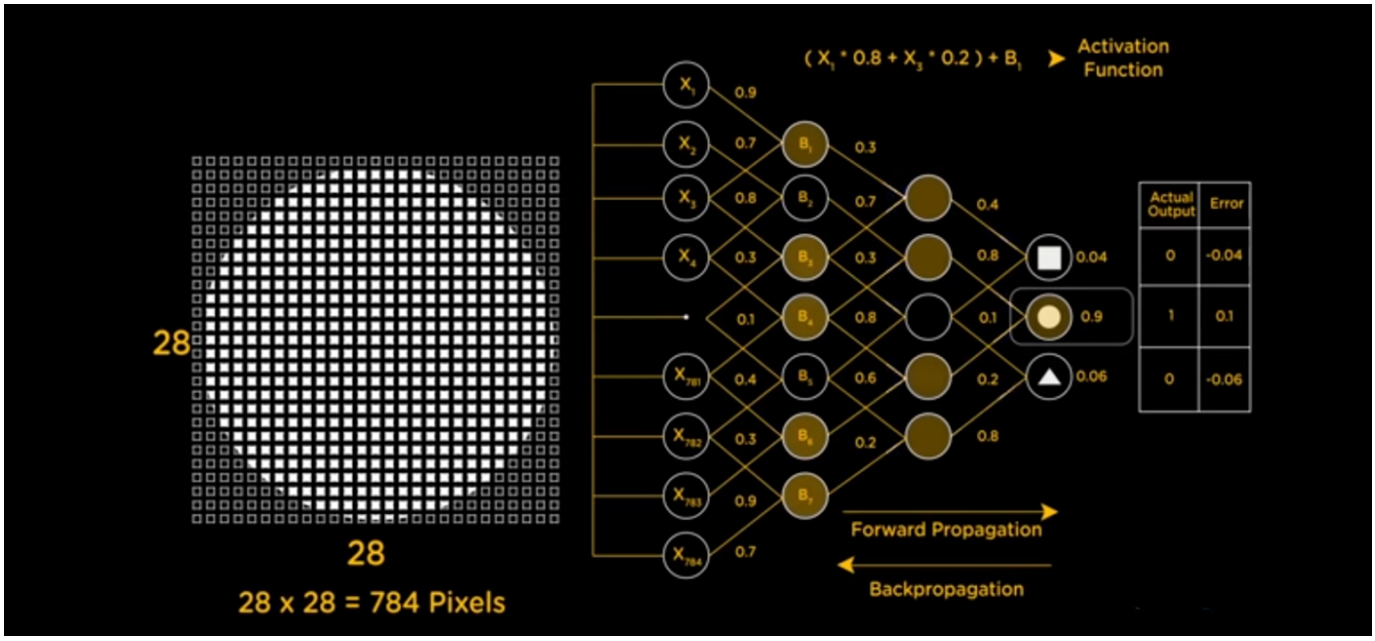
Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. Visual diagram of an input



Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

Let's try to understand neural networks with an example. Consider a situation in which the task is to recognize the given shape. Let consider the shape to be distributed into 28*28 pixels which makes up for 784 pixels and is fed as an input to each neuron of the first layer. Neurons of the first layer are connected to neurons of the next layer through channels and each of these

channels has its own value known as Weight. This value is then passed through a threshold function called the Activation function. The result of the activation function determines if the particular neuron is activated or not.



An activated neuron transmits data to the neurons of the next layer over the channels and this is how the data is propagated through the network and this is called Forward propagation. In the output layer the neuron with the highest value determines the output. These values are basically a probability.

Our network also has the output fed to it. The predicted output is then compared with the actual output to calculate the error in the prediction. The magnitude of the error indicates how wrong we are and the sign suggests if our predicted values are higher or lower than expected. This information is then relocated backward through our network. This is known as Back propagation. Based on this information the weights are adjusted. This cycle of forward propagation and backpropagation is iteratively performed with multiple inputs. The process goes on until we get appropriate weights that produce correct output and predict our shape correctly.

BackPropagation: The main feature of backpropagation is its iterative, recursive and efficient method for calculating the weights updates to improve the network until it is able to perform the task for which it is being trained. This was a basic idea and theory behind how Artificial Neural Networks actually work.

Now, it's obvious to think that what's Deep Learning and how's it different from Neural Networks, so let us clarify it as it's important to keep that in mind.

Now let's see what is the difference between Deep Learning and Neural Networks?

Watch the video

The obvious difference between Deep Learning and Neural Networks can be that Neural Networks operates similar to neurons in the human brain to perform various computation tasks faster while Deep Learning is a special type of machine learning that imitates the learning approach humans use to gain knowledge.

Deep Neural Network (DNN) is an artificial neural network with multiple layers between input and output layers. Each neuron in one layer connects to all the neurons in the next layer. The one or more layers between input and output layers are called hidden layers.

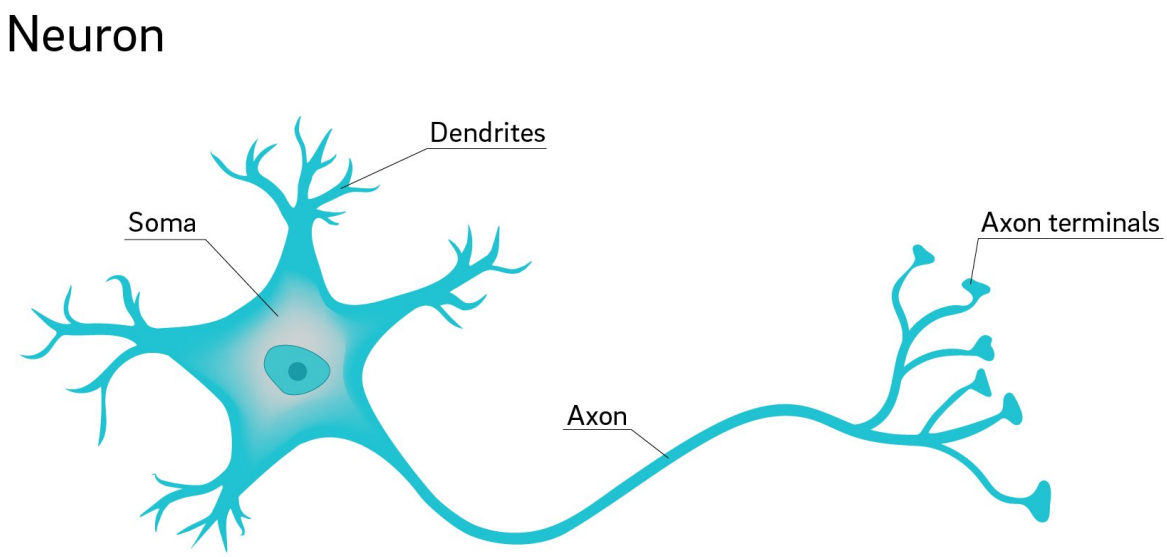
Building Blocks of Neural Networks

There are different types of neural networks but they always consist of the same components: layes, neurons, synapses, weights, biases, and functions.

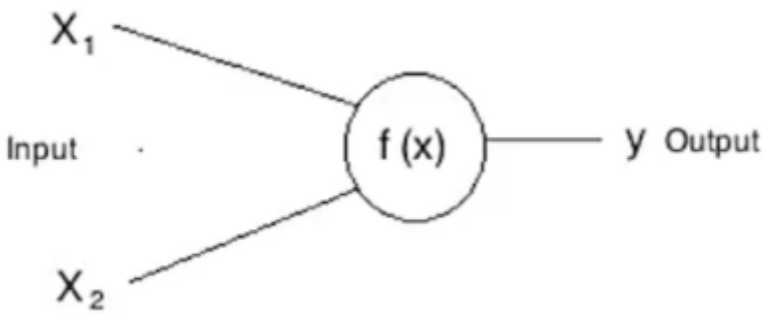
Neurons

A neuron or a node is a basic unit of neural networks that receives information, performs simple calculations, and passes it further. A layer consists of small individual units called neurons. A

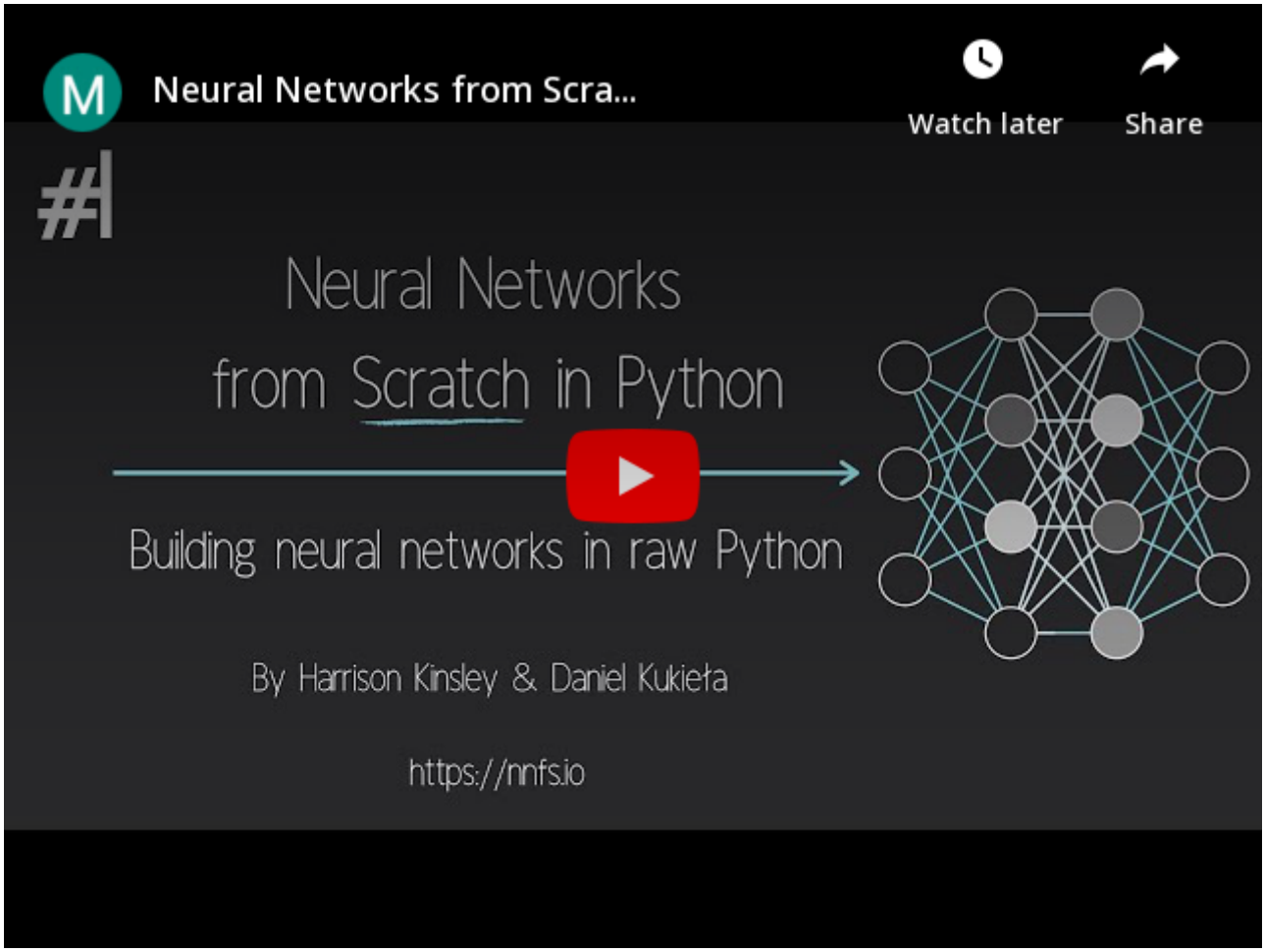
neuron in a neural network can be better understood with the help of biological neurons. An artificial neuron is similar to a biological neuron. It receives input from the other neurons, performs some processing, and produces an output.



Now let's see an artificial neuron



Here, X1 and X2 are inputs to the artificial neurons, f(X) represents the processing done on the inputs and y represents the output of the neuron.



A Single Neuron

Let's say we have a single neuron, and there are three inputs to this neuron. As in most cases, when you initialize parameters in neural networks, our network will have weights initialized randomly, and biases set as zero to start. Why we do this will become apparent later on. The input will be either actual training data or the outputs of neurons from the previous layer in the neural network. We're just going to make up values to start with as input for now

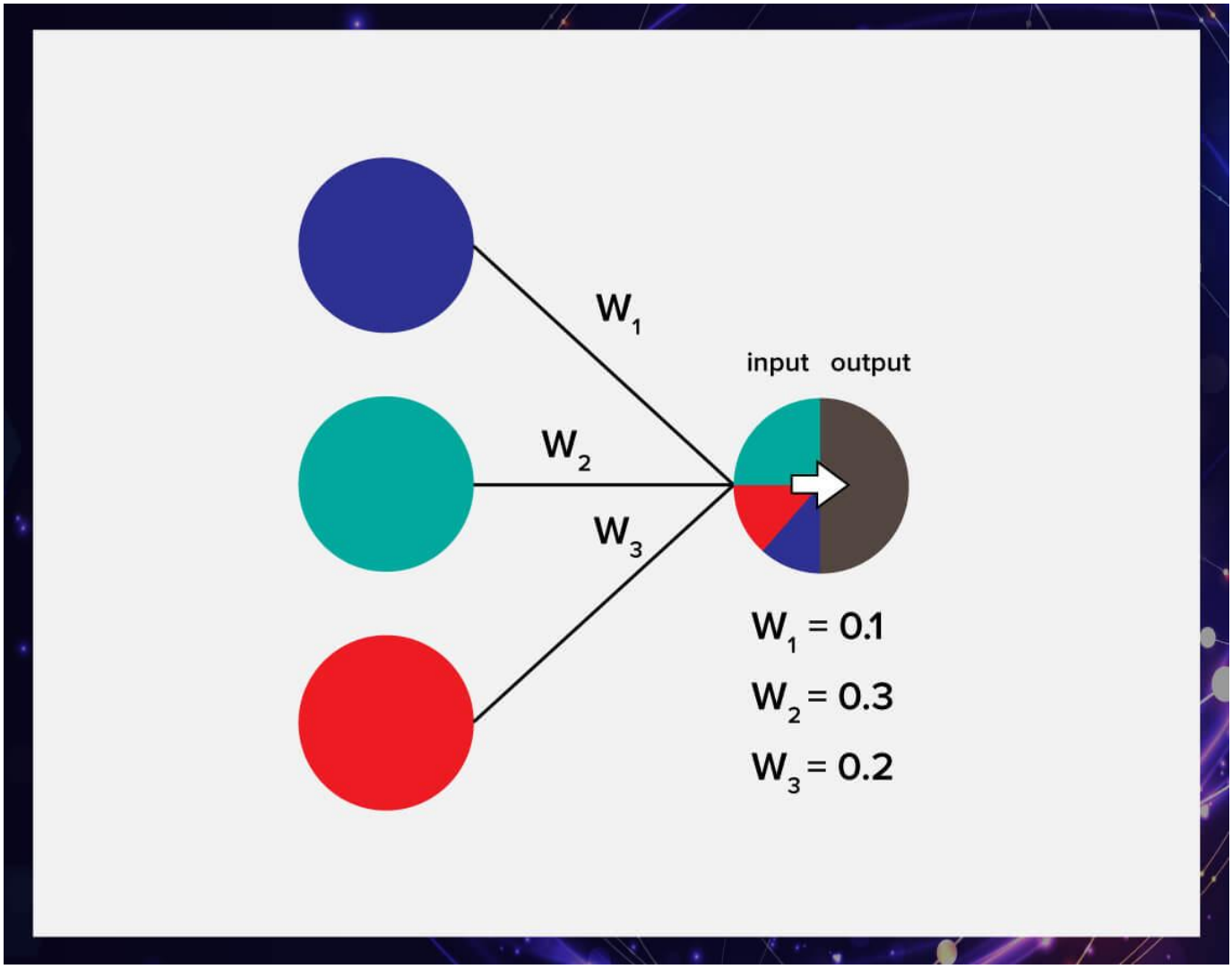
```
#create a variable named inputs which will be equal to list and pass the values 1,2,3 in t
```

```
#print the inputs
```

```
    [1, 2, 3]
```

Synapses and weights

A synapse is what connects the neurons like an electricity cable. Every synapse has a weight. The weights also add to the changes in the input information. The results of the neuron with the greater weight will be dominant in the next neuron, while information from less 'weighty' neurons will not be passed over. One can say that the matrix of weights governs the whole neural system



How do you know which neuron has the biggest weight? During the initialization (first launch of the NN), the weights are randomly assigned but then you will have to optimize them.

Now, let's get back to code

So as we already know that each input also needs a weight associated with it. **Inputs** are the data that we pass into the model to get desired outputs, while the **weights** are the parameters that we'll tune later on to get these results. **Weights** are one of the types of values that change inside the model during the training phase, along with biases that also change during training. The values for weights and biases are what get "trained," and they are what make a model actually work (or not work). We'll start by making up weights for now. Let's say the first input, at index 0, which is a 1, has a weight of 0.2, the second input has a weight of 0.8, and the third input has a weight of -0.5. Our input and weights lists should now be:

```
# create a variable named weights, which will be equal to list and pass the values which h
```

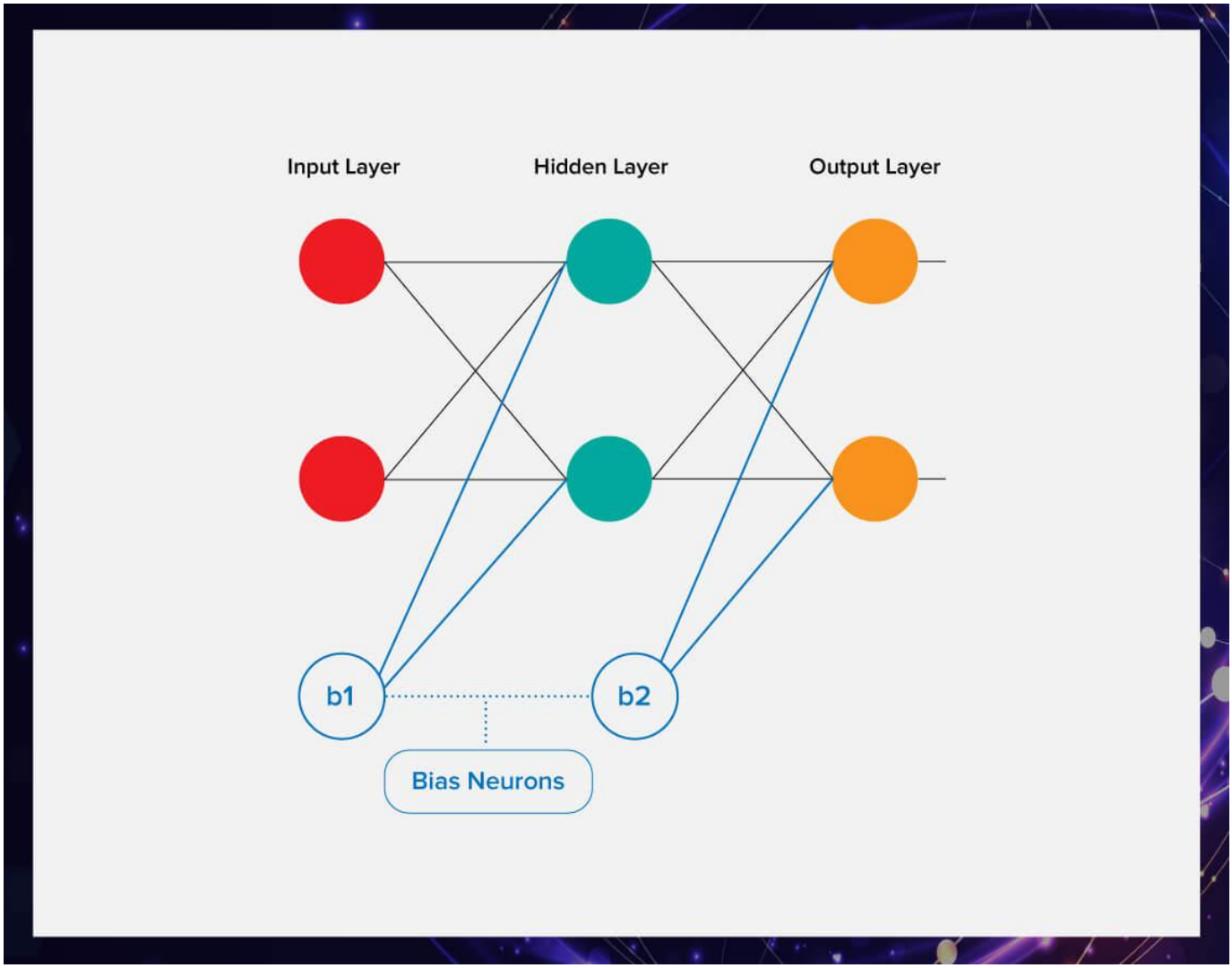
```
# print the weights
```

```
[0.2, 0.8, -0.5]
```

Bias

A bias neuron allows for more variations of weights to be stored. Biases add richer representation of the input space to the model's weights.

In the case of neural networks, a bias neuron is added to every layer. It plays a vital role by making it possible to move the activation function to the left or right on the graph.



It is true that ANNs can work without bias neurons. However, they are almost always added and counted as an indispensable part of the overall model.

Now as we know that we need the bias, At the moment, we're modeling a single neuron with three inputs. Since we're modeling a single neuron, we only have one bias, as there's just one bias value per neuron. The bias is an additional tunable value but is not associated with any input in contrast to the weights. We'll randomly select a value of **2** as the bias for this example:

```
#create a variable named bias which will be equal to 2
```

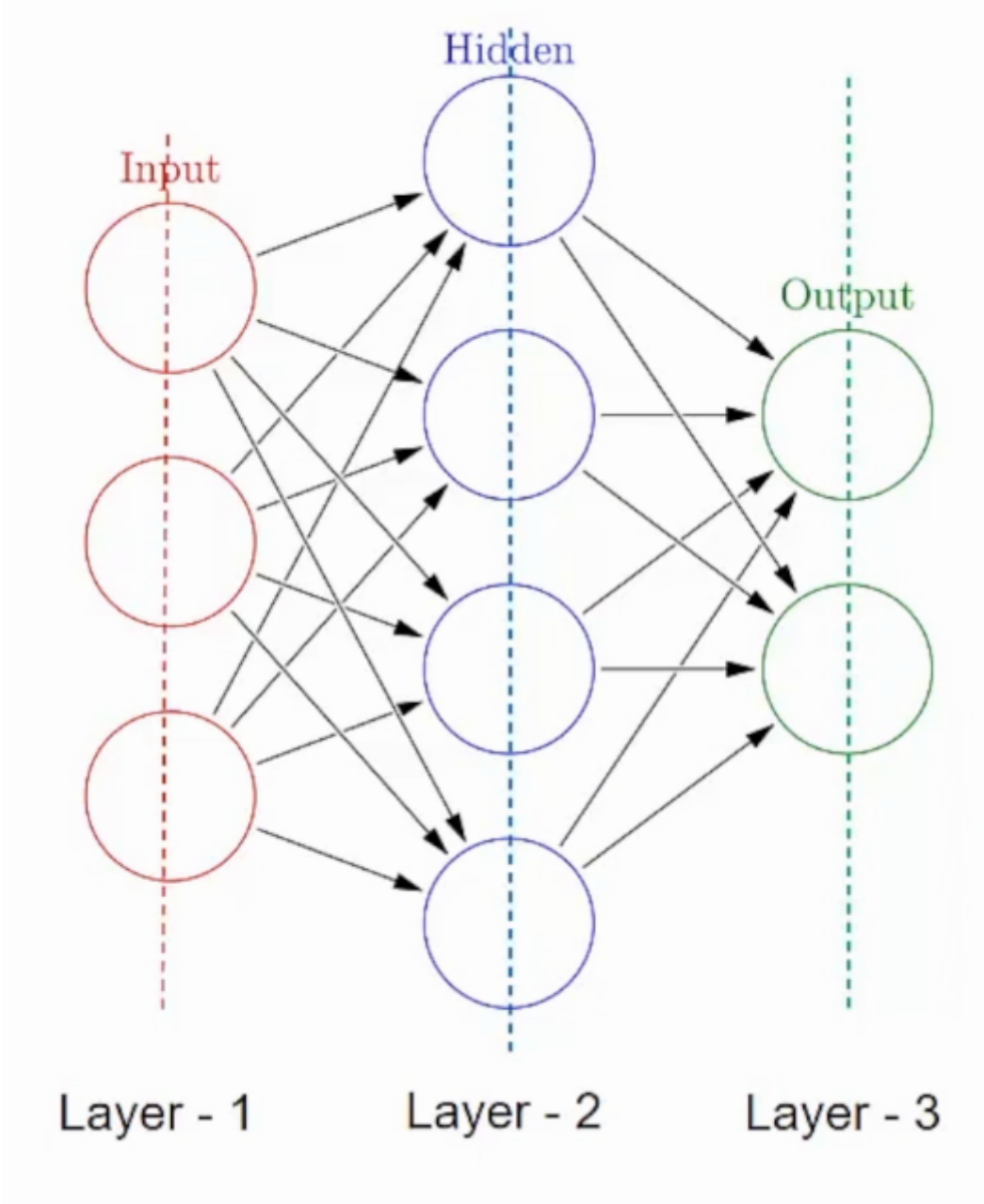
Layers

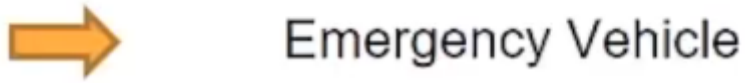
A neural network is made up of vertically stacked components called Layers. Each dotted line in the image represents a layer. There are three types of layers in a Neural Network.

Input Layer – First is the input layer. This layer will accept the data and pass it to the rest of the network.

Hidden Layer – The second type of layer is called the hidden layer. Hidden layers are either one or more in number for a neural network. In the above case, the number is 1. Hidden layers are the ones that are actually responsible for the excellent performance and complexity of neural networks. They perform multiple functions at the same time such as data transformation, automatic feature creation, etc.

Output layer– The last type of layer is the output layer. The output layer holds the result or the output of the problem. Raw images get passed to the input layer and we receive output in the output layer. For example


Hidden
Input
Output
Layer - 1
Layer - 2
Layer - 3



In this case, we are providing an image of a vehicle and this output layer will provide an output whether it is an emergency or non-emergency vehicle, after passing through the input and hidden layers of course.

Now, let's get back to our code

The neuron basicsally sums each input multiplied by that input's weight, then add the bias.

All the neuron does is take the fractions of inputs, where these fractions (weights) are the adjustable parameters, and adds another adjustable parameter — the bias — then outputs the result. Our output would be calculated up to this point like:


```
output = (inputs[0]*weights[0]+ inputs[1]*weights[1]+inputs[2]*weights[2]+ bias)
```

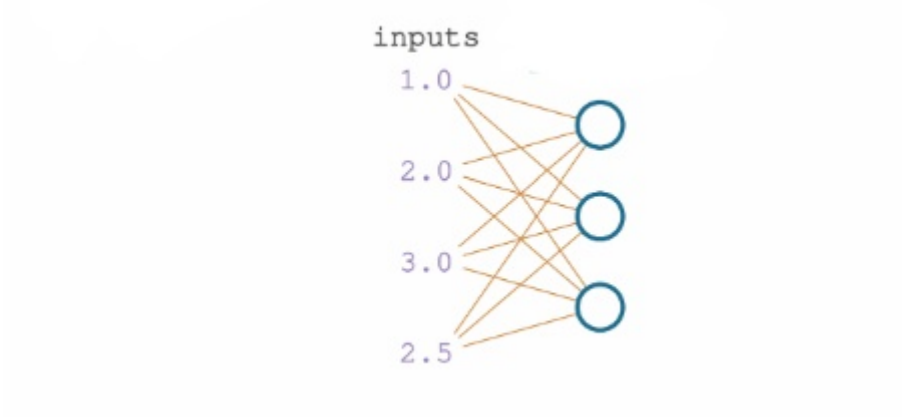
See the above image and code accordingly

```
# create a variable named output


# print the output



    2.3
```

So the output here we got is **2.3**

You did this for a single neuron. Now, let's do the same for a **Layer of Neurons**

Neural networks typically have layers that consist of more than one neuron. **Layers are nothing more than groups of neurons**. Each neuron in a layer takes exactly the same input — the input given to the layer (which can be either the training data or the output from the previous layer), but contains its own set of weights and its own bias, producing its own unique output. The layer's output is a set of each of these outputs — one per each neuron. Let's say we have a scenario with 3 neurons in a layer and 4 inputs.



We'll keep the initial 4 inputs and set of weights for the first neuron the same as we've been using so far. We'll add 2 additional, made up, sets of weights and 2 additional biases to form 2 new neurons for a total of 3 in the layer.

NOTE

> 1. Input layer will be having 4 values in the list, see the above image and mention the same values in the list of the input layer.
> 2. There will be 3 weights and each weight will be having four values inside the list.
>   - Consider **0.2, 0.8, -0.5, 1** as the values of the first weight.
>   - Consider **0.5, -0.91, 0.26, -0.5** as the values of the second weight.

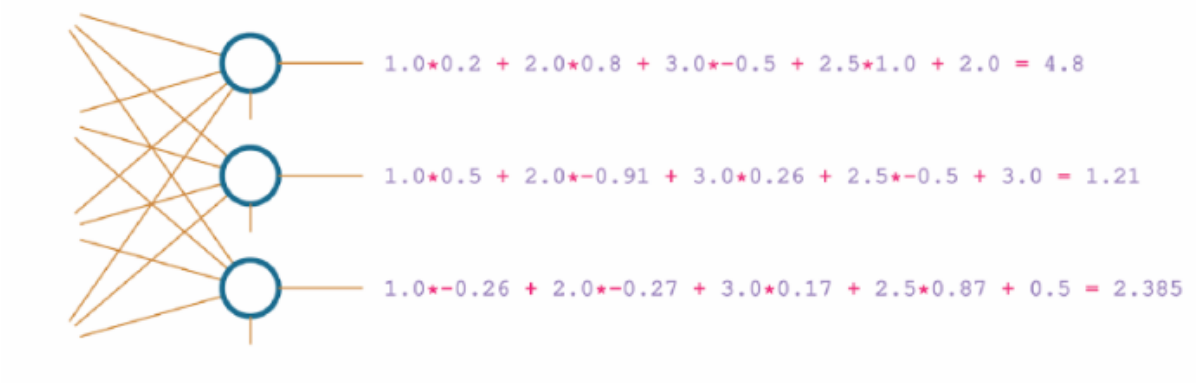
- Consider **-0.26, -0.27, 0.17, 0.87** as the values of the third weight.

3. Ther will be three biases:
    - Consider **2** as the value of first bias
    - Consider **3** as the value of the second bias
    - Consider **0.5** as the value of the third bias

```
# create a variable named inputs and store the values in the list as mentioned above




# Create a variable named weights1 and store the value as mentioned above

# Create a variable named weights2 and store the value as mentioned above

# Create a variable named weights3 and store the value as mentioned above


# create a variable named bias1

# create a variable named bias2

# create a variable named bias3
```

Now, let's code for the output layer

The layer's output is going to be a list of 3 values because we are having 3 neurons. Since there are three neurons, so each and every input layer will be getting multiplied with every weights and bias will be added accoridngly.




```
# create a variable named outputs

outputs = [
        # Neuron 1:
        inputs[0]*weights1[0] +
        inputs[1]*weights1[1] +
        inputs[2]*weights1[2] +
        inputs[3]*weights1[3] + bias1,

        # Neuron 2:
        inputs[0]*weights2[0] +
        inputs[1]*weights2[1] +
        inputs[2]*weights2[2] +
        inputs[3]*weights2[3] + bias2,

        # Neuron 3:
        inputs[0]*weights3[0] +
        inputs[1]*weights3[1] +
        inputs[2]*weights3[2] +
        inputs[3]*weights3[3] + bias3]

print(outputs)

```

See the above image and code in the below cell in a similar way.

```
# create a variable named outputs

        # Neuron 1:

        # Neuron 2:

        # Neuron 3:
```

```
# print the outputs
```

```
    [4.8, 1.21, 2.385]
```

So as you can we have got the list with 3 values which is basically the values of the **output layer**

Now we have done everything from scratch but this can be made more simmpler with a magical library called **NUMPY**

A Single neuron with **NumPy**

Let's code the solution, for a single neuron to start, using the dot product and the addition of the vectors with NumPy. This makes the code much simpler to read and write (and faster to run)

```
# import the library
```

```
# create a variable named inputs which will be equal  to the list with four values in it
# put 1,2,3,2.5 in the list of inputs
```

```
# create a variable named weights which will be equal to four values in the list.
# put 0.2,  0.8,  -0.5,  1.0 in the list of weights
```

```
# create a variable named bias whis will be equal to 2
```

So in the above cell we have created a list of values for the input layer, then we also have a list of values for the weights, and we also have a bias, now its time to multiply the input layer with the weights and the end we need to add the bias but we will do this by using the **dot()** function of numpy.

The **dot()** basically function returns the dot product of two arrays. For 2-D vectors, it is the equivalent to matrix multiplication.

```
outputs = np.dot(weights, inputs) + bias
```

See the above image and code the same in the below cell.

```
# create a variable named outputs which will be equal to the dot function on numpy
# inside the dot function pass weights and inputs and at the end add the bias



# print the output
```

```
    4.8
```

So here we got **4.8** and if you go up in the assignment where we calculated the output of single neuron from scratch we were getting the same output.

A Layer of Neurons with NumPy

Now we're back to the point where we'd like to calculate the output of a layer of 3 neurons, which means the weights will be a matrix or list of weight vectors. In plain Python, we wrote this as a list of lists. With NumPy, this will be a 2-dimensional array, which we'll call a matrix. Previously with the 3-neuron example, we performed a multiplication of those weights with a list containing inputs, which resulted in a list of output values — one per neuron.

We have done the same thing from scratch but now we are going to use **NumPy** here.

So let's code for three neurons

Note that the values in the list of input layer will be 1.0, 2.0, 3.0, 2.5

```
# create a variable named inputs and pass the above mentioned values



# print the inputs
```

```
[1.0, 2.0, 3.0, 2.5]
```

Now its time to mention the weights, which will be basically a list of lists and each list will be having four values

---

Note A list of lists in Python is a list object where each list element is a list by itself.

```
weights = [[0.2, 0.8, -0.5, 1],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
```

See the above image and code in the below cell accordingly.

```
# create a variable named weights and pass the values as mentioned above
```

```
# print the weights
```

```
[[0.2, 0.8, -0.5, 1], [0.5, -0.91, 0.26, -0.5], [-0.26, -0.27, 0.17, 0.87]]
```

Now as you know the next step is to mention bias.

NOTE: Consider 2.0, 3.0, 0.5 as the values in the list of bias

```
# create a variable named biases which will be equal to list
# pass the values which are mentioned above, inside the list
```

```
# print the biases
```

```
[2.0, 3.0, 0.5]
```

So as you know the next step is to calculate outputs, for that you need to use the **dot()** function of numpy.

```
# create a variable named output which will be equal to the dot function of numpy
# inside the dot() function pass weights, inputs and at the end add biases
```

```
# print the output
```

```
[4.8   1.21  2.385]
```

It is clearly seen that you got the same output which you got when you did this from scratch.

That's all for today, have a break and then move to the next assignment

Congratulations you have learned about the building blocks of neural network

Do Fill the feedback form !!

Feedback Form