A Quick Dive into Deep Learning

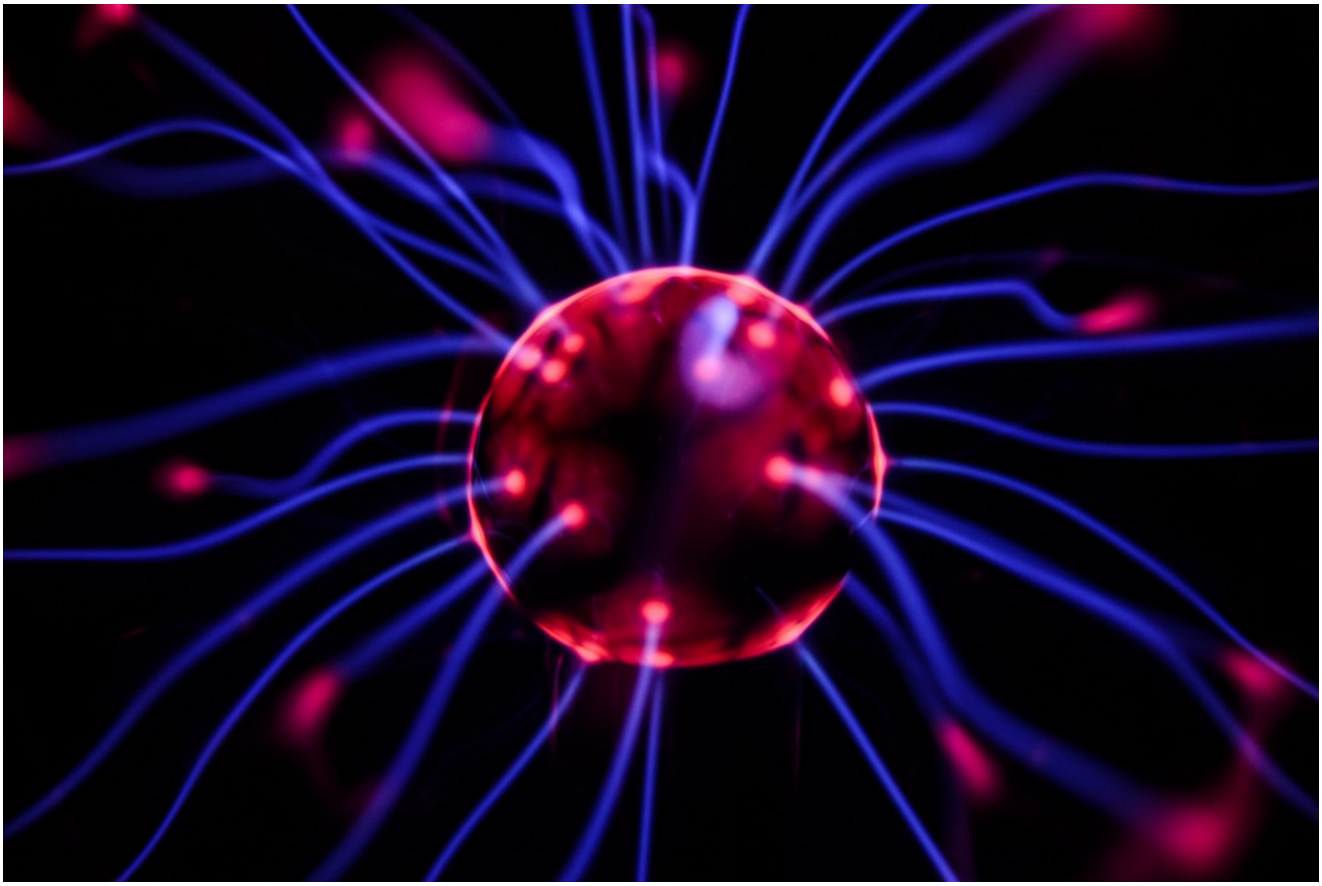

Photo by on

Deep learning is a popular and rapidly growing area of machine learning. Deep learning algorithms are a family of machine learning algorithms that use multi-layer **artificial neural networks** (ANNs) to perform classification tasks. An artificial neural network is a network of **artificial neurons**, loosely modeled after a network of animal neurons. An artificial neuron takes a series of inputs (here, x_1 through x_n), usually assigning each input a **weight**. It sums them, passing the sum through some type of non-linear function. Then it produces an output (here, *y)*.

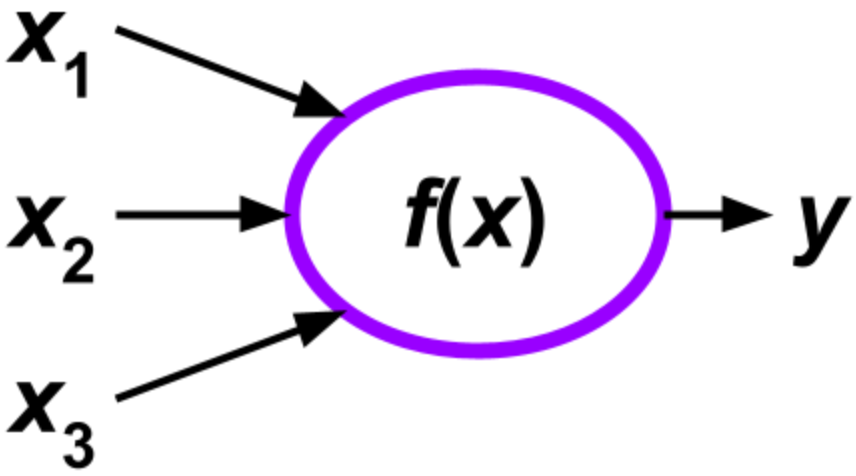

Diagram of an artificial neuron. Image mine.

In an artificial neural network, the artificial neurons are very specifically ordered in layers. Neurons in each layer are only connected to neurons in adjacent layers. A neuron in a layer can receive inputs from many other neurons in the previous layer and can pass its output to many other neurons in the next layer. The first layer of a neural network, taking input from outside, is called the **input layer**. The last layer, sending the final output, is called the **output layer**. Between the input layer and the output later are one or more **hidden layers**.

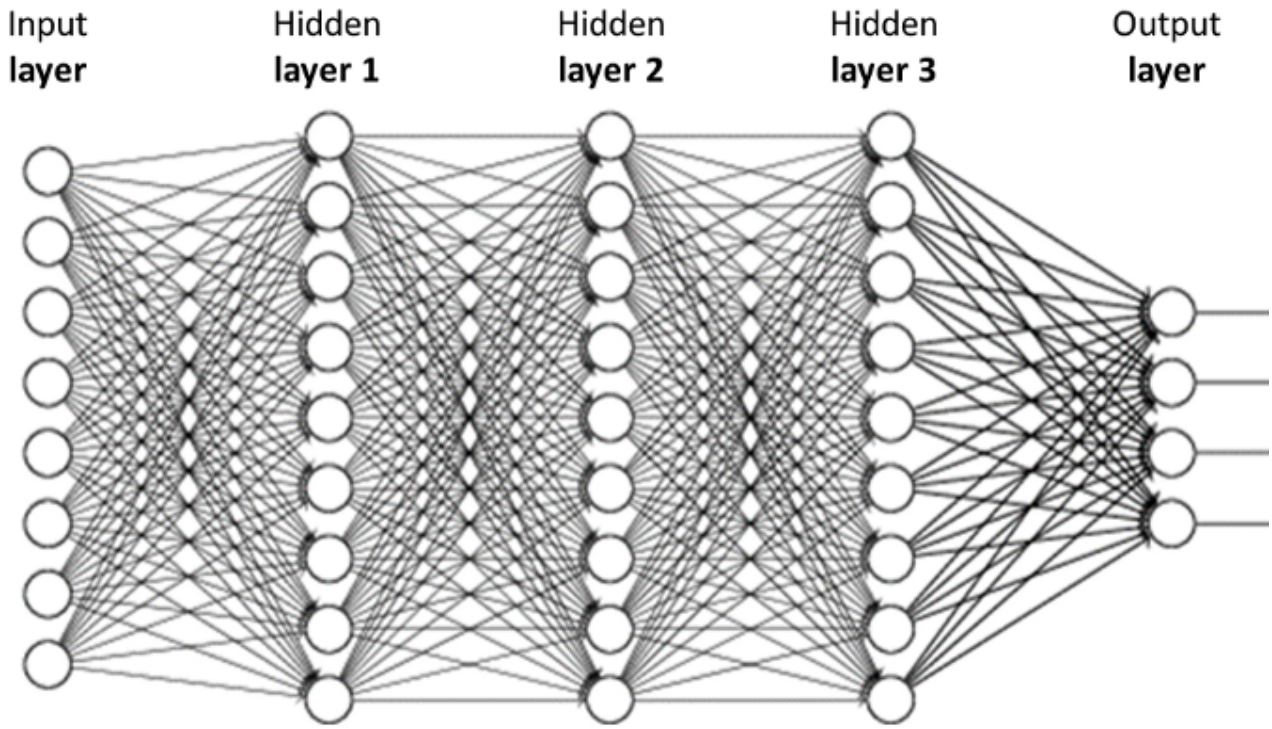

Diagram of a multilayer neural network. Image from M. Nielsen, Neural Networks and Deep Learning (2019)

So now, we'll do our first hands-on exercise to demonstrate how artificial neural networks are implemented. This artificial neural network will be very, very simple — much simpler than if it were used in a real-world application. In fact, it will be made up of only one neuron, with three inputs and one output. Open a Jupyter notebook in Python 3, and away we go.

First, we describe what the neuron would be like when it is created. The three inputs are stored in a 3 x 1 matrix and assigned random weights. The weights are adjusted after training.

```
from numpy import array, dot, exp, randomclass Neuron():    def __init__(self):
random.seed(15)        self.weights = 2 * random.random((3, 1)) - 1
```

These weighted inputs will be passed through a **sigmoid** function, which produces an S-shaped curve. Put the code for the sigmoid function in class Neuron, along with the code for the sigmoid function's derivative — the gradient of the sigmoid curve, a measure of our confidence about the existing weight.

```
def __sigmoid(self, x):    return 1 / (1 + exp(-x))def __sigmoid_derivative(self, x):
return x * (1 - x)
```

To run the neural network, we pass our inputs through it, allowing them to be transformed by the sigmoid function. The method to run the neural network also belongs in class Neuron.

```
def run(self, inputs):    return self.__sigmoid(dot(inputs, self.weights))
```

And finally, we train the neural network by running it through many iterations, calculating the discrepancy between the predicted and actual output value (the error), and adjusting the weights accordingly. The method for training the network is the final method to go into class Neuron.

```
def train(self, training_inputs, training_outputs, iterations):    for iteration in
range(training_iterations):        output = self.run(training_inputs)        error =
training_outputs - output        confidence = error *
self.__sigmoid_derivative(output)        adjustment = dot(training_inputs.T,
confidence)        self.weights += adjustment
```

So here is the full class Neuron. Make sure you properly indent each method and loop — Python is one of the few programming languages where the interpreter will not even be able to read code without the proper indentation.

```
class Neuron():    def __init__(self):        random.seed(15)        self.weights = 2
* random.random((3, 1)) - 1    def __sigmoid(self, x):        return 1 / (1 + exp(-
x))

    def __sigmoid_derivative(self, x):
        return x * (1 - x)

        def run(self, inputs):        return self.__sigmoid(dot(inputs,
self.weights))    def train(self, training_inputs, training_outputs, iterations):
for iteration in range(iterations):            output = self.run(training_inputs)
error = training_outputs - output            confidence = error *
self.__sigmoid_derivative(output)            adjustment = dot(training_inputs.T,
confidence)            self.weights += adjustment
```

Train the neural network by running it 10,000 times, as follows.

```
if __name__ == "__main__":    network = Neuron()    print("Random starting weights:
")    print(network.weights)    inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0,
1, 1]])    expected_outputs = array([[0, 1, 1, 0]]).T    network.train(inputs,
expected_outputs, 10000)    print("New weights after training: ")
print(network.weights)    # Test the neural network with new input.    print("Test of
input [1, 0, 0] -> ?: ")    print(network.run(array([1, 0, 0])))
```

You should print the following:

```
Random starting weights: [[ 0.69763539] [-0.64220815] [-0.89127357]]New weights after
training: [[ 9.67341006] [-0.20852727] [-4.62943754]]Test of input [1, 0, 0] -> ?:
[0.99993707]
```

Photo by on

Congratulations! You have just built and run your very first neural network. I hope you enjoyed your first foray into the world of deep learning, and you want more.